Basic structure

The structure of the scene file is organized as a declaration of the camera, lights, subgraphs, and the main or root scenegraph. Both the root scenegraph and subgraph declarations are in the form of a tree. Each node of the tree can contain any number of children which can be drawn in any order. Each child consists of a transformation block (the order of the transformations *within* the block does matter) and either an actual object (sphere, cone, *etc.,*) or a previously declared subgraph. Please note that subgraphs must be named. The general format of the file is as follows:

```
camera declaration
light declarations (up to 8)
named subgraph declarations (any number, plus one called root)
```

The camera and light declarations look like this:

```
camera [
      eye <vector>
      look/focus <vector>
      up <vector>
      angle <angle>
      ]

light [
      type <lighttype>
      id <number>
      position <vector>
      color <color>
      function <color>
      direction <vector>
      aperture <number>
      exponent <number>
      ambient <color>
      ]
```

The scene graph declarations are recursive and are the same for both the subgraphs and the root subgraph. Each node of a tree takes the form of a transformation block (any number of matrix operations, maybe none) followed by the object the transformation is applied to. An object can be one of the following: a standard object (cone, sphere, etc.) or a previously defined and named subgraph. This is the format for a translation block:

```
      trans [
            rotate <vector> angle
            translate <vector>
            scale <vector>
            matrixRC <matrix>
            matrixCR <matrix>
            .
            object to apply transformation to
      ]
```

This is the form of a subgraph with the two types of sub-trees; an object or a previously defined subgraph. (The root subgraph is identical to this except that the root label indicates that this is the root node.) Remember that a subgraph (or sub-tree) can have multiple transformation blocks within it.

```
subgraph name [
      trans [
            . transformations
            object object [
                  diffuse <color>
                  ambient <color>
                  reflect <color>
                  specular <color>
                  shine index
                  transparent <color>
                  ior index
                  texture filename u v
                  ]
      ]
      trans [
            . transformations
            subgraph name of previously declared subgraph
      ]
      trans [
            . transformations
      ]
]
```

The root subgraph is indicated by using the name `root`.

The italicized types are defined as follows:

- *vector* — three floating point numbers representing a displacement, axis of rotation, scaling factors, or coefficients.
- *angle* — a single floating point number representing an angle, in degrees.
- *matrix??* — 16 floating point numbers representing a matrix in row-major (matrixRC) or column-major order (matrixCR).
- *lighttype* — one of the keywords `point`, `directional`, or `spotlight`
- *color* — three floating point numbers representing a color in the red-green-blue color space.
- *index* — a single floating point number representing some index or coefficient
- *name* — a unique string representing the name of the master scenegraph. Can be used to instantiate a copy of the master scenegraph later on.
- *object* — a string representing an object type. Can be one of: cube, cylinder, cone, sphere, or cow (optional).
- *filename* — the name of the file containing the bitmap used as a texture map for the object. If an absolute path name is not given, and the texture can't be found in the current directory. The u and v parameters are doubles that indicate how many times the texture should repeat (may by 2, 0.5, etc.).

Block details

Camera:

There can only be one camera in the scene, and if it is present, it will be the first thing listed in the scene. The default camera is described below. In a camera block, there must be listed the eye-point (`eye`),

look vector (`look`), up vector (`up`), and height angle (`angle`). Note that it is possible to replace the look vector with a focal point (denoted by the keyword `focus`), if so desired.

Lights:

It is necessary for the light to have a particular type: point, directional, or spotlight. If none is given, the light type will default to a point light. There can be up to 8 lights in the scene. The id of a particular light is specified by the `number` parameter (indexed from 0 to 7). There are two parameters that must be defined for each point light. The first, `position`, is the actual position of the light in the scene. Directional lights will not have a position — they will have a vector `direction`. Spotlights, or cone shaped lights need both `position` and `direction`, as well as `aperture`. The second parameter, `color`, is the color of the light, in rgb. Lastly, there is a parameter, `function`, that allows one to specify the coefficients of the falloff function. The three arguments are the coefficients of the constant, linear, and quadratic term, respectively.

The ambient light term sets the ambient light; the last ambient line parsed is what the ambient light will be.

Transformations:

Each object in the scene must be immediately surrounded by a transformation block. The first portion of a transformation block is a list of the transformations that will be applied to the object within. This list can be empty, though such is usually the case for only the top level transformation. This list can consist of any number of rotations, translations, scalings, and matrix multiplications. They will be composed in the order they are listed, meaning that the last transformation type listed will be the first to be applied to the enclosed object. The last part of a transformation block is the object itself. There can be only one object, and nothing should follow the object. If anything is found after the first object in a translation block, it should be ignored.

Objects:

There are two kinds of object primitives — instances of previously declared subgraphs and object primitives. The first type, `subgraph` allows you to reuse an already defined subgraph, the name of which is specified in the *name* field. Primitives are those which represent three-dimensional shapes, namely, `cube`s, `cylinder`s, `cone`s, `sphere`s, and the cow mesh (optionally). Primitives contain in their block a combination of surface characteristic definitions: diffuse color `diffuse`, ambient color hack `ambient`, reflected color `reflect`, specular color `specular`, specular exponent `shine`, transparency `transparent`, and index of refraction `ior`. There is also the ability to provide texture to objects using a bitmap. This behavior is added using the `texture` keyword with the name of the file containing the bitmap as well as coefficients for the scaling of the texture.

Quick Reference

Camera:

Keyword:	Description:	Arguments:	Example:
`eye`	The camera's eyepoint	<*vector*>	`eye 1 2 3`
`look`	The camera's look vector	<*vector*>	`look 1 2 3`
`focus`	The camera's focal point	<*vector*>	`focus 1 2 3`
`up`	The camera's up vector	<*vector*>	`up 1 2 3`
`angle`	The camera's height angle in degrees	<*angle*>	`angle 60`

If more than one camera is present, all but the first should be ignored. If no camera is present, the default is for the camera to be defined as the following:

```
camera [
   eye 0 0 1
   look 0 0 -1
   up 0 1 0
   angle 60
   ]
```

Lights:

Keyword:	Description:	Argument
`type`	The light's type	<*type*>
`id`	The light's id	<*index*>
`position`	The light's position	<*vector*>
`direction`	The light's direction	<*vector*>
`aperture`	The spotlight's cone angle (degrees) **double**	
`exponent`	The spotlight's exponential falloff (positive number, bigger numbers are sharper) **double**	
`color`	The light's color	<*color*>
`function`	The falloff function	<*vector*>

Transformations:

Keyword:	Description:	Arguments:	Example:
`rotate`	A rotation about a vector originating at the origin	<*vector*>, *angle*	`rotate 1 -2`
`translate`	A translation	<*vector*>	`translate 1`
`scale`	A resizing	<*vector*>	`scale 1 2 3`
`matrixRC, matrixCR`	An arbitrary transformation matrix	<*matrix*> in row-column or column row order	

Objects: (*All objects are centered at the origin*)

Object:	Description:
`cube`	A cube with unit length edges
`cylinder`	A cylinder that extends one unit above and below the XZ plane, and has a radius of .5 units.
`cone`	A cone that extends one unit above and below the XZ plane, and has a radius of 0 at `0 1 0`, and a radius of .5 at `0 -1 0`.
`sphere`	A sphere with radius .5.

Surface qualities:

Keyword:	Description:	Arguments:	Default:
`ambient`	The ambient color	<*color*>	`ambient 0 0 0`
`diffuse`	The diffuse color	<*color*>	`diffuse 1 1 1`
`reflect`	The reflected color	<*color*>	`reflect 0 0 0`
`specular`	The specular color	<*color*>	`specular 0 0 0`
`shine`	The specular exponent	<*index*>	`shine 1`

`transparent`	The transparency	<*color*>	`transparent 0 0 0`
`ior`	The index of refraction	<*index*>	`ior 1`
`texture`	The texture map	<*filename*> <*u*> <*v*>	`texture wood.ras 2 4`
`emit`	The emission color (for path tracing only)	<*color*>	`emit 0 0 0`

Notes

This format is not as all-encompassing as it could be. This is intentional, as one of the major aims is for it to be simple to parse. Also noteworthy is the fact that much of what the file format supports, namely most of the surface characteristics, will not be used until later assignments, and so can be initially ignored, though provisions should be made for easy handling later on.