The CSCI1230 Scene File Format

Basic structure

The structure of the scene file is organized as a declaration of the global data, camera, lights, master objects/sub-scenegraphs, and special master object 'root'/main scene graph. Both the main scene graph's and sub-scenegraphs' declarations are in the form of a tree. All sub-scenegraphs must be declared before the main scene graph. Within both the main scene graph and the sub-scenegraphs, a given subtree can contain any number of tranformation blocks which can be executed in any order, since they only affect the object contained in the tranformation block. The objects can either be actual physical objects that are represented in the scene, or a subtree that would inherit the transformation for all its children. Scenes are specified in xml format. The general format of the file is as follows:

```
<scenefile>
    <globaldata>
        <diffusecoeff float/>
        <ambientcoeff float/>
        <transparentcoeff float/>
        <specularcoeff float/>
    </globaldata>

    <cameradata>
        <pos vector/>
        <focus/look vector/>
        <up vector/>
        <heightangle float/>
    </cameradata>

    <lightdata>
        <id number/>
        <type lighttype/>
        <position vector/>
        <color color/>
        <function vector/>
        <direction vector/>
        <penumbra float/>
    </lightdata>

     <object type="tree" name=name>
        <transblock>
            <rotate vector angle/>
            <translate vector/>
            <scale vector/>
            <matrix matrix/>
            ⋮
```

```
            object tree name
            ⋮
        </transblock>
        <transblock>
            ⋮
            <object type="primitive" object=object>
                <diffuse color/>
                <specular color/>
                <ambient color/>
                <reflect color/>
                <transparent color/>
                <shine float/>
                <blend float/>
                <texture float filename u v/>
                <bumpmap float/>
            </object>
            ⋮
        </transblock>
        ⋮
    </object>
</scenefile>
```

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 order
- *lighttype* — one of the keywords for light types
- *color* — three floating point numbers representing a color in the red-green-blue (values 0-1) color space
- *index* — a single floating point number representing some index or coefficient
- *name* — a unique string representing the name of the object. Can be used to instantiate the object later on. Omitting the name indicates that the object will not be instantiated again. Master objects/sub- scenegraphs must have names. The name root should be reserved for the main scene graph. The name root signifies that all the master objects/sub-scenegraphs have been declared.
- *object* — a string representing an object type. Can be one of: cube, cylinder, cone, sphere, mesh or tree. If it is mesh, then the filename of the mesh file must be the

next attribute in the form `filename="<filename>"` Its properties are specified like a primitive's.

- *filename* — the name of the file containing the bitmap used as a texture map for the object

Block details

Global Data:

You can leave this section out. It allows the scene file writer to specify certain constants used while rendering. Each of the coefficients has a certain place in the lighting equation. They are all floating point constants that are used to scale the diffuse, ambient, transparent, and specular components respectively. These constants are helpful for scenes with very few or very many lights.

Camera:

There can only be one camera in the scene. If none is specified a default camera will be used. In a camera block, there must be listed the position (`pos`), 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, spot or area. If none is given, the light type will default to a point light. You will only deal with non-point lights in Ray. (In this assignment, treat all lights as if they were point lights. — this should make your life easier.) 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 `penumbra`. Spotlights will be extra credit — speak to a TA if you wish to implement them. 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 lighting function. The three arguments are the coefficients of the constant, linear, and quadratic term, respectively.

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 four kinds of object primitives, which can be divided into three distinct types ? groupings of objects, instances of master objects, and object primitives. The first type, "groupings," consists solely of the `tree` object, which simply contains any number of transformation blocks. The second, "instances," allows you to reuse an already defined master object, the name of which is specified in the *name* field. Primitives are those which represent three-dimensional shapes, namely, `cubes`, `cylinders`, `cones`, `spheres`, and `meshes`.

Primitives contain in their block a combination of surface characteristic definitions: diffuse color `diffuse`, ambient color hack `ambient`, reflected color `reflective`, specular color `specular`, specular exponent `shininess`, transparency `transparent`, and index of refraction `ior`. There is also the ability to provide texture to object 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. This is also the ability to specify a bump map file.

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.