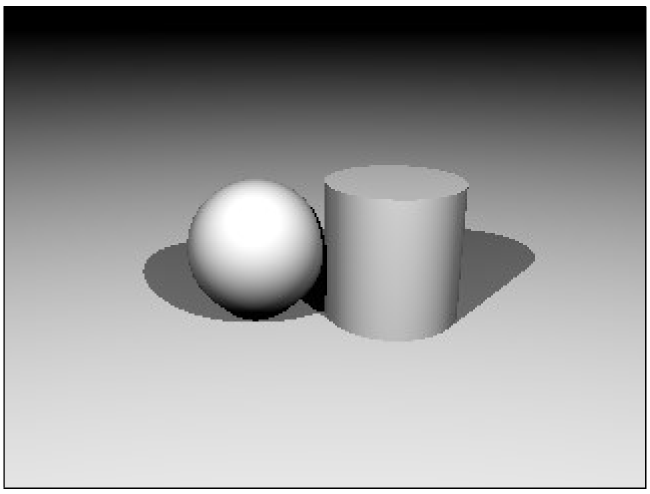
ITC303 Software Engineering - Assignment 3

JTRACER USER MANUAL

Version 0.1

Joel Sheehan (11334071)



CONTENTS

VERSION HISTORY

4/11/07 – Initial version completed

JTRACER OVERVIEW

JTracer is a simplistic ray tracing system developed for ease of use. JTracer supports the rendering of sphere, plane and cylindrical primitive objects, with the use of both point and ambient lights. Surface properties such as ambient, diffuse, specular and reflective properties may be defined for individual primitives, while the primitives themselves may be translated, rotated and scaled within world space.

The greatest power of JTracer comes with its simple scene description language, which allows the user to control every element of the JTracer system from within a single scene description file. This eliminates the need for the user to learn and remember complicated command line switches for controlling features of the program.

PROGRAM EXECUTION

In an effort to lessen the learning curve required to use JTracer as much as possible, command like options for program execution have been eliminated from the system in favour of options defined within a scene description file. To produce a render of a scene it is as simple as executing JTracer specifying a scene description file filename as the first argument to the program.

EXAMPLE:

```
jtracer scene_01.jts
```

Scene Description File

A scene description file is the main source of input for JTracer. Scene description files can specify the entirety of the information required to produce a render of a scene.

To correctly specify a scene, information should be specified within `render_settings`, `cameras` and `scene` definition blocks. These definition blocks control over the render system, viewpoints and scene composition respectively, providing all the necessary information to render a scene.

JTracer Scene Description Files themselves are simply plain text files with the extension '.jts', which stands for "JTracer Scene". The file extension is not checked at runtime however, and any extension may be used.

Numbers

Numeric values within a scene description file follow normal decimal format, with an optional minus sign at the beginning of a number, and a sequence of decimal digits. The number may then optionally specify a decimal point followed by an additional sequence of decimal digits in order to represent real numbers.

Examples:

```
-1
0.5
3.14159
```

Vectors and Colours

Vectors within a JTracer Scene Description File are denoted simply as a sequence of three numbers, separated from each other using commas, surrounded by angle brackets.

A colour vector is simply a vector which has its numeric values clipped between zero and one in order to represent an RGB triple. That is, the first number in the vector represents the red component of the colour, the second number represents the green component, and the third the blue.

By definition the coordinate system used by JTracer is a right-handed system, with the z-axis protruding out of the screen when the x-axis points to the right, and the y-axis upward, with the origin located at the bottom left hand corner of the screen.

EXAMPLES:

```
<0,0,-4>
<0.0, 0.0, 0.0>
<1.0, 9.1, .3.2>
```

STRINGS

Strings are represented in the JTracer Scene Description Langauge as a sequence of characters surrounded by double quotation marks. There are two restrictions on the character sequence composing a string. The first restriction is that the character sequence can only contain characters in the printable ASCII character range. The second restriction is that a character sequence cannot contain double quotation marks. Strings are used to label cameras, and provide an output filename to JTracer.

EXAMPLES:

```
"Camera One"
"RedSpheres.bmp"
```

COMMENTS

'End of Line' comments can be placed anywhere in a scene description file by denoting the start of a comment using the '#' character. After a # character is read in by the JTracer parsing system, any further input is ignored until the end of the line is found. Currently there are no multi-line comments implemented.

EXAMPLES:

```
# this is a comment
# and this is a separate one
```

RENDER SETTINGS

JTracer has a range of settings that can be changed to alter the properties of the image to be rendered. The following sections outline each property and their uses. Render settings are specified within a `render_settings` block in the scene description file. Multiple `render_settings` blocks may be defined within a file, however only the last block will be used.

WIDTH AND HEIGHT

The `width` and `height` properties can be used to specify the dimensions of the image to be rendered. The values specified must be integers greater than zero. Values will be truncated to the integer component if decimals are specified. The default value for both properties is zero.

EXAMPLE:

```
render_settings {
    width: 640;
    height: 480;
}
```

RECURSE_DEPTH

The `recurse_depth` property is used to limit the recurse depth for secondary rays. By default this property is set to zero, so it should be specified explicitly to render reflections. Values specified must be positive integers. Values specified with fractional components will be truncated to the integer component.

EXAMPLE:

```
render_settings {
    recurse_depth: 5;
}
```

FILENAME

The `filename` property is used to specify the name of the file to output the rendered image to. The file extension is ignored in terms of output, and all files are output in

Windows Bitmap format. String values must be specified for the `filename` attribute, and if no `filename` is specified, output will be written to "jtracer.bmp" in the current working directory.

EXAMPLE:

```
render_settings {
    filename: "scene_01.bmp";
}
```

ACTIVE_CAMERA

The `active_camera` property is used to specify the name of the camera that will be used as the viewpoint to render the scene from. This allows multiple cameras to be defined within a scene description file, and facilitates rapid testing of multiple viewpoints. If no camera name is specified, or the name does not match an existing camera the first camera defined is used.

EXAMPLE:

```
render_settings {
    active_camera: "Far Angle Camera";
}
```

BACKGROUND

The `background` property specifies the background colour for the rendered scene. This is the colour that is drawn when there are no objects to be drawn. This property is required to be defined using a colour vector.

EXAMPLE:

```
render_settings {
    # a magenta background
    background: <1.0, 0.0, 1.0>;
}
```

AMBIENT_LIGHT

This property specifies the intensity of the ambient light. By default this light is black, but can be specified by this property. This property is defined using a colour vector.

EXAMPLE:

```
render_settings {
    # faint red ambient light
    ambient_light: <0.2, 0.0, 0.0>;
}
```

SETTING CAMERAS

Every render of a scene must have at least one camera defined to provide a viewpoint and orientation from which to view the scene. This is done by creating a camera definition within the `cameras` block in the scene description file. The `cameras` block acts as a container for a selection of cameras, to facilitate the definition of multiple viewpoints and minimise the work required to switch viewpoints for renders.

PERSPECTIVE_CAMERA

The `perspective_camera` block provides definitions of perspective projection cameras. Perspective cameras provide genuine depth cueing effects for natural looking images.

NAME

The `name` attribute allows a name to be associated with this camera, enabling it to be designated as the current viewpoint using the `active_camera` render setting. The `name` attribute is a string attribute.

EXAMPLE:

```
perspective_camera {
    active_camera: "Far Angle Camera";
}
```

POSITION, LOOK_AT AND UP

The `position`, `look_at` and `up` vectors define all the necessary information to position and orient a `perspective_camera`. The `position` attribute defines the position of the camera, the `look_at` attribute defines the direction that the camera is looking in, and the `up` attribute defines the upward direction for the camera. The `up` vector can be thought of as the direction of the top side of a physical camera. Each of these attributes are vector attributes.

EXAMPLE:

```
perspective_camera {
    # a perspective camera looking just above
    # the origin, with the upward direction along
    # the x-axis.
    position: <5.0, 5.0, 5.0>;
    look_at: <0.0, 1.0, 0.0>;
    up: <1.0, 0.0, 0.0>;
}
```

ASPECT

The `aspect` property facilitates the control of aspect ratios during image synthesis. Typically this ratio will be the width of the rendered image divided by the height of the image (See the `width` and `height` render settings). The `aspect` property has the value 1 by default so explicit definition of the aspect ratio is often required. The aspect ratio is a numeric property.

EXAMPLE:

```
perspective_camera {
    # 4:3 aspect ratio
    aspect: 1.333333333;
}
```

FOV

The `fov` property designates the vertical field of view of the perspective camera in degrees. `fov` is a numeric property, and is set to 45 degrees by default.

EXAMPLE:

```
perspective_camera {
    # 60 degree vertical field of view
    fov: 60.0;
}
```

SETTING THE SCENE

Definition of primitive objects and light sources is a must in ray tracing to be able to synthesis an image of anything at all. The `scene` block contains the definitions for all primitives and light sources to be rendered. Multiple `scene` blocks may be defined within a scene description file, however only the last block will contain the scene to be rendered.

EXAMPLE:

```
scene {
    sphere {
        center: <1.0, 0.0, 0.0>;
    }
    point_light {
        position: <5.0, 5.0, 3.0>;
        intensity: <0.8, 0.8, 0.0>;
    }
}
```

GEOMETRIC PRIMITIVES

Geometric primitives are objects that can be arranged in a manner to create a representation of the scene to be rendered.

SPHERE

The `sphere` primitive provides a representation of a sphere that can be centred on any point in space, with any positive radius. It can also be transformed to produce spheroids and ellipsoids.

CENTER

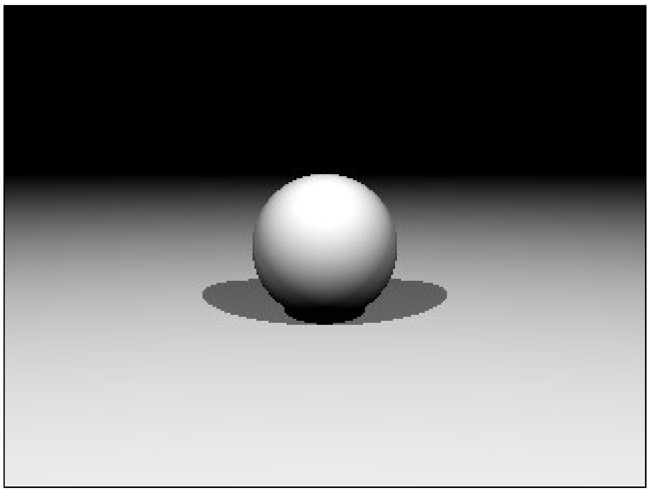
The `center` attribute for a `sphere` is a vector attribute that defines the centre-point for the `sphere`. By default a `sphere`'s centre is the origin.

RADIUS

The `radius` attribute for a `sphere` defines the `sphere`'s radius. By default all `spheres` have a `radius` of 1 unit, though a `radius` can be specified as any positive number.

EXAMPLE:

```
sphere {
    # a sphere of radius 2, translated along the y-axis
    center: <0.0, 2.0, 0.0>;
    radius: 2.0;
}
```



PLANE

The `plane` primitive creates a plane surface oriented in a particular direction, at a particular distance from the origin. As such, a `plane` is defined using a surface normal, and a distance. Planes may also be transformed to alternate locations and orientations.

NORMAL

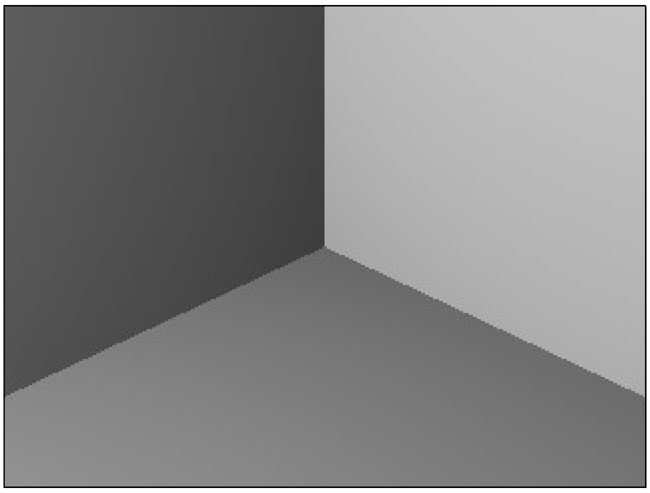
The `normal` property of a `plane` specifies the direction of the `plane`'s surface normal. The `normal` property is a vector, pointing along the y-axis by default.

DISTANCE

The `distance` property of a plane specifies how far from the origin the plane is located along its surface normal. By default this distance is zero, though planes may be specified at any distance from the origin.

EXAMPLE:

```
plane {
    # the plane z = -2
    normal: <0.0, 0.0, 1.0>;
    distance: -2.0;
}
```



CYLINDER

The `cylinder` primitive creates a circular cylinder oriented along the y-axis with end caps at y = 0 and at a height defined by the user. Cylinders can then be transformed to alternate locations and orientations, or to produce elliptical cylinders.

RADIUS

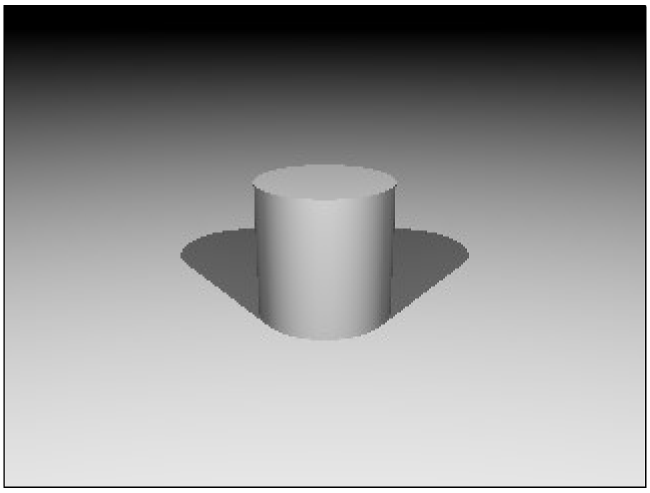
The `radius` property of a cylinder defines the `radius` of circular cross-section of the `cylinder`. By default a `cylinder` has a `radius` of half a unit, but may be specified as any non-zero positive number.

HEIGHT

The `height` property of a `cylinder` defines the height of the cylinder along the y-axis, with respect to the bottom of the cylinder at y = 0. By default the `height` of a `cylinder` is 1 unit, but may be specified as any non-zero positive number.

EXAMPLE:

```
cylinder {
    # a cylinder of radius 0.75 units,
    # 5.0 units high from the origin
    radius: 0.75;
    height: 5.0;
}
```



LIGHT SOURCES

Light sources provide a number of crucial elements in a rendered scene, including specular highlights and diffuse reflection. These effects give a scene a realism that cannot be replicated using an ambient light.

POINT_LIGHT

Point lights are infinitesimally small light sources that radiate light equally in all directions. Point lights can be defined by specifying a position, and the intensity of the light that is radiated from the light source.

POSITION

The `position` attribute of a point light defines the position of the point light in world space. By default this vector property is set to the origin.

INTENSITY

The `intensity` attribute defines the intensity of the light radiated by the light source. This property accepts colour vectors, and by default is set to black.

EXAMPLE:

```
point_light {
    position: <1.5, 2.0, 4.0>;
    intensity: <0.0, 0.8, 0.0>;
}
```

MATERIALS

Materials are a set of properties that can be applied to primitive objects to change their colour and reflective properties. Materials are defined within a primitive's definition to change their properties. It should be noted that the properties defined within a material will not always combine to produce a result that is expected.

PIGMENT

The `pigment` property changes the colour of a primitive. This property accepts colour vectors and by default is set to black by default.

AMBIENT

The `ambient` property defines the ambient reflective colour of the primitive. This colour is combined with the ambient light's intensity, and the pigment of the primitive to produce the final ambient colour component of the primitive. This colour vector is set to black by default.

DIFFUSE, DIFFUSE_K

Diffuse reflection is a form of reflection where the intensity of light reflected decreases as the angle between the light source and the surface normal increases. The `diffuse` property defines the diffuse reflective colour of the light that is diffusely reflected from the primitive. By default the `diffuse` colour vector is defined to black.

The `diffuse_k` property is an exponential component of the diffuse reflection equation that controls the intensity of light as the angle between the light source and the surface normal increases. A greater `diffuse_k` will increase the intensity of reflected light at greater angles between the light source and surface normal. By default the `diffuse_k` property is set to 1, with potential values greater than zero.

SPECULAR, SPECULAR_K

Specular highlights are the perfect reflection of light from a light source from a surface, that is, when the angle between the surface normal and the eye is the same as the angle between the surface normal and the light. When this occurs, a small bright area of reflected light is created on the surface which is frequently witnessed on the surface of smooth objects.

The `specular` property of a material controls the colour of the specular highlight, while the `specular_k` property controls the size of the specular highlight. The `specular` property is a colour vector that is set to black as a default, while the `specular_k` property is a positive number set to 64 as a default. The greater the value of `specular_k`, the smaller the size of the specular highlight will be.

REFLECTION

The `reflection` property of a material defines the percentage of light incident on a surface that is reflected perfectly off the surface. This property can be used to create mirror reflection effects on surfaces. By default this property is set to zero, but can be assigned values between zero and one.

EXAMPLE:

```
sphere {
    # a sphere that is
    material {
        pigment: <0.8, 0.8, 0.8>; # almost white
        ambient: <0.0, 0.0, 0.0>; # no ambient reflect
        diffuse: <0.0, 0.8, 0.0>; # green diffuse light
        diffuse_k: 1.0;
        specular: <0.8, 0.0, 0.0>; # red highlights
        specular_k: 128.0;         # small highlights
        reflection: 1.0;       # and highly reflective
    }
}
```

TRANSFORMS

Transformations may be applied to primitive objects in order to change their shape, orientation or position in space. Transforms are applied to a primitive by specifying a transform within the definition block of that primitive, with each transform type being applied in the order specified in the transform definition block.

TRANSLATE

The `translate` transform moves a primitive around in world space relative to the vector supplied. A vector `<5, 0, 0>` will move a primitive object 5 units along the x-axis.

SCALE

The `scale` transform scales a primitive along the x,y and z axes according to each component of a vector assigned to the property.

XROTATE, YROTATE, ZROTATE

The `xrotate`, `yrotate` and `zrotate` transforms rotate a primitive about the x, y and z axes respectively, by an angle defined in degrees as designated.

EXAMPLE:

```
cylinder {
    radius: 0.5;
    height: 1.0;
    transform {
        translate: <0.0, -0.5, 0.0>; # translate downward
        zrotate: 45.0;           # rotate about z-axis
        scale: <2.0, 0.0, 0.0>;  # scale along x-axis
    }
}
```