1

Shape:

This is an abstract class used to describe all objects which one can intersect with a ray. This includes the basic, Sphere, Triangle, Box, Cylinder, Cone and Line, but also includes higher level objects like Group and RayFileInstance.

The most important method that all these objects share is the **intersect** method. When this method is called the object determines the closest point of intersection of the ray with the object. If the ray does intersect the object then the ray fills in the intersection-information object passed in and returns the distance along the ray to the point of intersection. Otherwise it returns -1.0.

2

IntersectionInfo:

This structure is used to store information about the point of intersection. In particular, it stores:

- The material properties of the point of intersection
- The coordinates of the point of intersection (object coordinates)
- The normal to the surface at the point of intersection
- The texture coordinates at the point of intersection

3

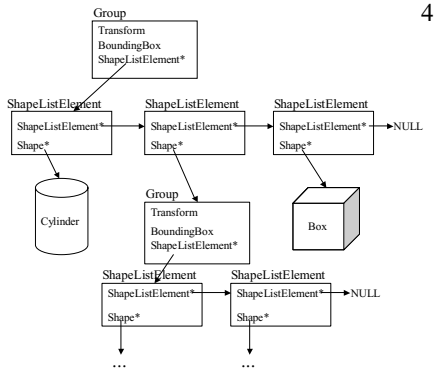
Group:

A node of the scene-graph. It contains a pointer to a ShapeListElement (a list of Shapes), an object-to-world transform that is to be applied to each of the child Shapes, and a bounding-box that contains all of the child shapes.

ShapeListElement:

A data structure used for storing a list of shapes. Each ShapeListElement has a pointer to a Shape and a pointer to the next ShapeListElement in the list.

4



5

RayFile:

When a .ray file includes shapes that are described in other .ray files this object is used to store the information described in these secondary .ray files.

6

RayFileInstance:

Frequently one desires to use a complex shape multiple times in the same scene (at different positions, orientations, scalings, etc). This can be done by generating a separate .ray file describing the shape and then adding the scene-graph of the complex shape into the initial scene-graph (as a node).

Instead of having multiple copies of the same complex shape in multiple positions, this object serves as a container to the corresponding RayFile (i.e. it stores a pointer to the RayFile object). Any intersection querries on the complex shape are then redirected to the RayFile. (Which is to say, *RayFileInstance.intersect* does the same as *RayFileInstance.rayFile.scene.intersect.*)

Light: 7

This is the abstract class used to describe all lights. It provides methods for determining the specular and diffuse contributions of a light source at a point of intersection, and it provides a method for determining if the intersection point is in shadow with respect to the light.

The three light types that are implemented are:

- Point Light: A light with a fixed position in space.
- Spot Light: A light with a fixed position in space and a restricted cone within which the light falls.
- Directional Light: A light with a specified direction but no fixed position in space.

The strength of both the spot-light and point-light attenuates as you move further from light source.

Scene: 8

This object stores all of the information read out from the .ray file. This includes:

- Ambient and Background colors
- Camera
- The number and list of lights
- A pointer to the root node of the scene-graph.

Additionally it has (as private members)

- The number and list of Textures/Materials/include RayFiles/Vertices

Camera: 9

This object stores information about the camera within the scene. It stores:

- The half-height-angle
- The position of the camera
- The forward/up/right directions of the camera

These determine the position and direction of the view-frustrum

Vertex: 10

Triangles in the scene are defined by the vertices used. Since it is usually true that vertices get reused (in adjacent triangles), this object provides a way to store the vertices (and their normal and texture coordinate information) independent of the triangle. The triangle then is defined as three pointers to the three vertices that define it.

Material: 11

This object stores the information about the material properties of the object. This includes:

- The objects ambient/diffuse/specular/emissive color
- The specular coefficient of the object
- The transparent coefficient of the object
- The refraction index of the object
- The texture (if any) used to color the object.

Texture: 12

This object is used to store the texture information used for texture mapping an object. It contains the filename of the (.bmp) image and a copy of the image itself. (Use the tools provided in bmp.[cpp/h] to manipulate the image.)

Bounding Box: 13

In order to speed up ray-tracing the bounding box object is used to describe the "smallest" axis-aligned box containing the shape. Every Shape has a method that returns its bounding box, and the every Group stores the bounding box containing its children.

The bounding box itself is defined as a pair of points, with the first being the vertex of the box with smallest (x,y,z) values and the second being the vertex of the box with largest (x,y,z) values.

The Group's bounding box is obtained by calling the **getBoundingBox** method of each of its children, constructing the bounding box that contains all of the children's bounding boxes, and then transforming the bounding box into the Group's coordinate system (using the local transform).

Transforming the bounding box is done by transforming each of the vertices of the box and then computing a bounding box containing the new eight points.

Lighting: 14

Using the material properties the equation for the color of the light at a point of intersection is given by:

$$(\text{object.emmisive}) + (\text{object.ambient} \cdot \text{scene\_ambient}) +$$

$$\sum_{\text{lights}} (1 - \text{light.isInShadow}) \cdot (\text{light.getDiffuse} \cdot \text{object.diffuse}) +$$

$$\sum_{\text{lights}} (1 - \text{light.isInShadow}) \cdot (\text{light.getSpecular} \cdot \text{object.specular})$$

PointLight.isInShadow: 15

If the normal to the surface at the point of intersection is pointing away from the light (i.e. away from the direction from which the light is coming), the point is in shadow.

Otherwise you must generate the ray whose initial position is the point of intersection and whose direction is the direction of the line from the point to the light. Intersect the ray with the scene and determine if the ray hits anything before it hits the light. If it does then the point is in shadow, otherwise it is not.

SpotLight.isInShadow: 16

If the normal to the surface at the point of intersection is pointing away from the light (i.e. away from the direction from which the light is coming), the point is in shadow.

Otherwise you must generate the ray whose initial position is the point of intersection and whose direction is the direction of the line from the point of intersection to the light. Intersect the ray with the scene and determine if the ray hits anything before it hits the light.

Additionally you have to make sure that the angle between the line from the point of intersection to the light, and the direction of the spot light is less than the cut off angle.

If the ray hits an object in the scene before it hits the spot light, or if the angle is larger than the cut off angle then the point is in shadow.

DirectionalLight.isInShadow: 17

If the normal to the surface at the point of intersection is pointing away from the light (i.e. away from the direction from which the light is coming), the point is in shadow.

Otherwise you must generate the ray whose initial position is the point of intersection and whose direction is the negative direction of the light, (i.e it heads towards the light). Intersect the ray with the scene and determine if the ray hits anything. If it does then the point is in shadow.

Point Light (Attenuation): 18

If p is the point of intersection, c is the location of the light, ca is the constant attenuation, la is the linear attenuation, and qa is the quadratic attenuation, then the total attenuation of the light is given by:

$$\frac{1}{ca + la \cdot \|p - c\| + qa \cdot \|p - c\|^2}$$

19

Spot Light (Attenuation):

If p is the point of intersection, c is the location of the light, ca is the constant attenuation, la is the linear attenuation, qa is the quadratic attenuation, dor is the drop off rate, and α is the angle between the line from the point of intersection to the light, and the direction of the light, then the total attenuation of the light is given by:

$$\frac{\cos(\alpha)^{128.0 \cdot dor}}{ca + la \cdot \|p - c\| + qa \cdot \|p - c\|^2}$$

20

Light.getDiffuse:

If a is the attenuation of the light with respect to the point of intersection ($a=1.0$ in the case that the light is a directional light), α is the angle between the direction to the light and the surface normal at the point of intersection, and (r,g,b) is the color of the light then the diffuse color of the light reaching the point of intersection is:

$$(r \cdot a \cdot \cos(\alpha), g \cdot a \cdot \cos(\alpha), b \cdot a \cdot \cos(\alpha))$$

21

Light.getSpecular:

If $kspec$ is the specular coefficient at the point of intersection, α is the angle between the refracted ray direction and the direction from the point of intersection to the light, a is the attenuation of the light with respect to the point of intersection, and (r,g,b) is the color of the light then the specular color of the light reaching the point of intersection is:

$$\left(\cos(\alpha)^{128.0 \cdot kspec} \cdot a \cdot r, \cos(\alpha)^{128.0 \cdot kspec} \cdot a \cdot g, \cos(\alpha)^{128.0 \cdot kspec} \cdot a \cdot b\right)$$