Raja Developer's Manual

by Emmanuel Fleury, Grégoire Sutre

February 25, 2001

The Raja Project

Contents

Introduction

About this document

This document present some developer's needed informations about the structure of RAJA. This is **NOT** for beginner! We do not introduce the method of ray-tracing, neither the basic of Java programming. And we do not aim to do.

A very first approach

Ray-tracing is a concept which seems to be build for oriented object programming. Actually, this was not quite obvious in the very first time of Smalltalk, but we have now enough computing power to assume this fact.

The ray-tracing is a quite simple to define. Consider a point p in a 3-dimensional space and a screen s. The problem is to throws rays from p to s and consider their path refracting or reflecting on several solids to a light source.

In a classical programming way (C, Pascal, Fortran, ...), the ray-tracer must compute all rays, all intersections, all refractions, all reflection by himself, solids are only parameters of the problem. This way of coding is quite complex and the code is often unreadable (moreover if some optimizations have been included in it).

In an object oriented programming way, the code remain still unreadable. But, the part of the ray-tracer is less important. It is only a part of the code. Most of the calculus are done by solids themselves. A good way to do is to consider that all solids have a method which take a ray in parameter and return `null` if this ray don't intersect the solid or the intersection otherwise. Each solid can have his own method to compute this intersection and therefore his own optimization. Therefore, the ray-tracer is quite simple to understand because he only manages the throwing of the rays on the screen. Moreover, the code is more modular and we can easely add some new solids or ray-tracer (distributed ray-tracer).

Actually, all the complexity of the classic way of coding has not vanished. The problem is now to define a consistent and efficient hierarchy of classes, avoiding the problems caused by transparency, union, complement, intersection and so on.

We tried to do it for RAJA, and we about to think we have completed a big part of it!

Why in Java???

Why not! Actually, Java2 is the only object oriented language which provide a such complete API. Moreover, this language is more close of object oriented programming than C++. That's why we decided to code RAJA in Java. Recoding RAJA in C++ would be quite difficult because we strongly use the Java2 API and some others libraries. But, if somebody want to do it, we would support him.

Chapter 1

Class Hierarchy

This chapter aim to present and explain the hierarchy of classes and interfaces we have choosen to conceive RAJA.

We first introduce some general consideration on geometry classes (`Point3D` and `Vector3D`). Then we introduce lights and solids. Finally, we present the classes related to the raytracer and the parser.

1.1 Notations

During this document we will use some notations to make shorter the description of the classes:

- `Classe`: Simple Class,
- `Interface(Class)`: The type `Interface` is needed but, as it is an interface an occurence can only be provide by some implementation of this interface, `Class` is one of those,
- `Var = Class`: When there some ambiguity between variables and that the type can bring enough information to differentiate them,
- `List[Class]`: Objects list from the type `Class`.

We will present a class, like that:

Class Name
• Interface: - Interface(Implemented Class) • List[Class] • Var = Class

For example:

Scene
• Solid: - Solid(BasicSolid) - Solid(Aggregate) • List[LightSource] • backgroundLight = RGB • ambiantLight = RGB • ambiantVolume = RGB

The class `Scene` require an object `Solid` wich can be provide by the class `BasicSolid` or `Aggregate`. It also require a list of object `LightSource` and some `RGB` objects.

The following description of the hierarchy is not complete, because we focus only on describing the class hierarchy (for example we do not present methods). For more information you should look at the javadoc generated documentation and at the source code.

1.2 Geometry

This part gather all

Point3D
• x = Double • y = Double • z = Double

Vector3D
• x = Double • y = Double • z = Double

1.3 Lights

RGB
• r = Double • g = Double • b = Double

Texture
• kd = RGB • krl = RGB • krg = RGB • ktl = RGB • ktg = RGB • ns = Integer • nt = Integer

1.4 Solids

see fig. 1.1.

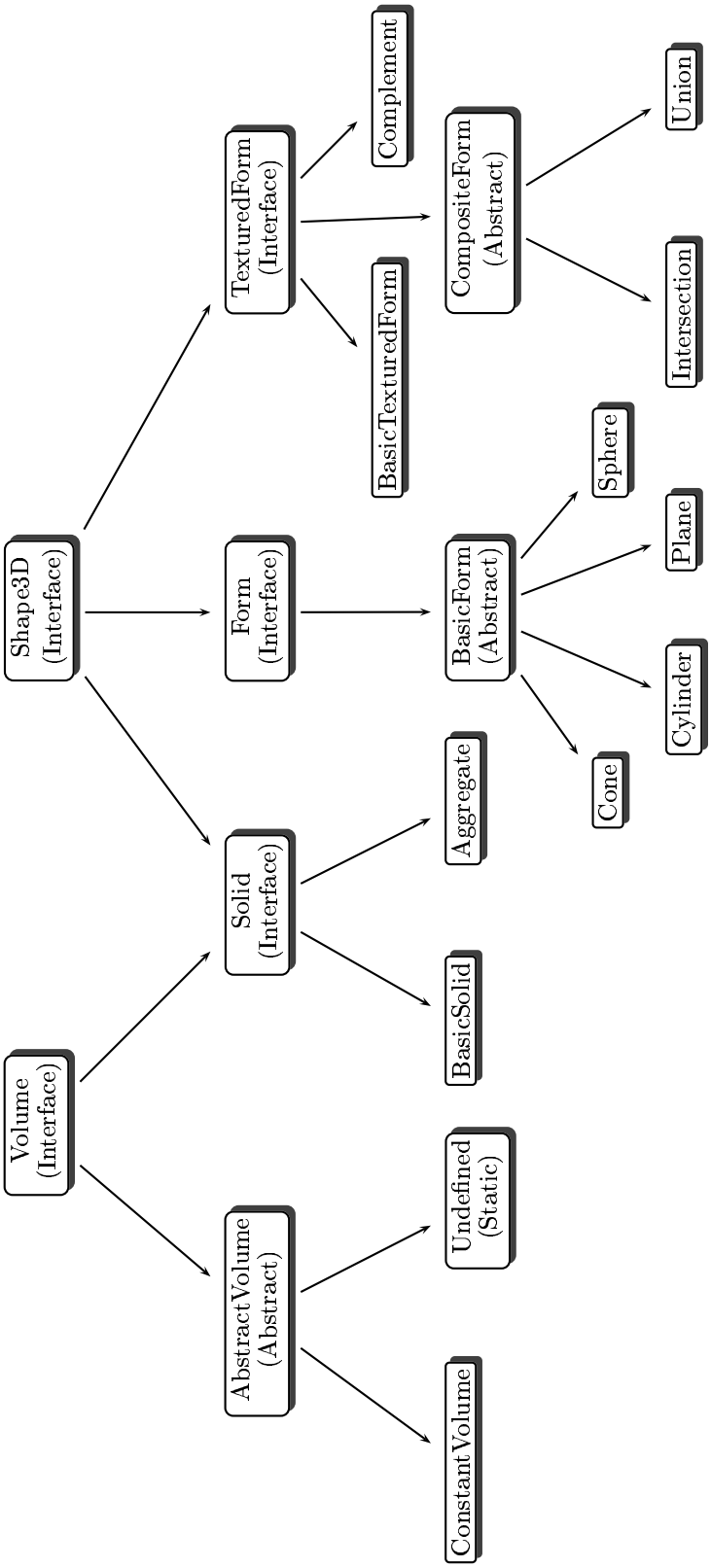

Figure 1.1: *Solids Class Hierarchy.*

1.4.1 Basic Classes

BasicSolid

- TexturedForm:
  - TexturedForm(BasicTexturedForm)
  - TexturedForm(Complement)
  - TexturedForm(Intersection)
  - TexturedForm(Union)
- Volume
  - Volume(ConstantVolume)
  - Volume(Undefined)

BasicTexturedForm

- Form:
  - Form(Cone)
  - Form(Cylinder)
  - Form(Plane)
  - Form(Sphere)
- Texture

Complement

- TexturedForm:
  - TexturedForm(BasicTexturedForm)
  - TexturedForm(Complement)
  - TexturedForm(Intersection)
  - TexturedForm(Union)

ConstantVolume

- refractiveIndex = Double

1.4.2 Composite Classes

Intersection

- List[Form]

Union

- List[Form]

Aggregate

- List[Form]
- SolidPriorityGraph

1.4.3 BasicForm Classes

Cone

- origin = Point3D
- direction = Vector3D
- angle = Double

Cylinder
• origin = Point3D • direction = Vector3D • angle = Double

Plane
• origin = Point3D • normal = Vector3D

Sphere
• center = Point3D • radius = Double

1.5 Raytracer

Scene
• Solid: - Solid(BasicSolid) - Solid(Aggregate) • List[LightSource] • backgroundLight = RGB • ambiantLight = RGB • ambiantVolume = RGB

La classe Scene

1.6 Parser

Chapter 2

Solids

2.1 What is a Solid?

2.2 Main methods

2.2.1 The method `intersect`

2.2.2 The method `build`

2.3 How to code a Solid?

2.3.1 Add a `BasicForm`

2.3.2 Add a `Texture`

2.3.3 Optimization Tips

Chapter 3

RayTracing Methods

3.1 Intersection

3.1.1 Overview

3.1.2 The Acne Problem

3.2 Reflection/Refraction

3.2.1 The Recursive Path

3.2.2 Reflection/Refration Formula

3.3 Sampling

3.3.1 Anti-aliasing

3.3.2 Diadic Sampler

3.4 Distributed Computation

3.4.1 Multi-threading

3.4.2 Networking

Chapter 4

Parser

4.1 Syntaxe

4.2 How does it work?

4.2.1 The class `Parser`

4.2.2 The method `build`

Chapter 5

Graphical User Interface

To be done!