ECE 462
Object-Oriented Programming using C++ and Java

Brief History of C++ and Java

Yung-Hsiang Lu
yunglu@purdue.edu

C++ History

- why to study history?
  - Knowing the past often helps us plan for the future.
  - The design decisions of one programming language help us design better languages.
- Since C++ (1982), many new programming languages have been developed:
  - 1991 Python
  - 1995 Java 1
  - 1995 PHP
  - 1997 OO COBOL
  - ...

History of C++: 1979-1991
by Bjarne Stroustrup

- background
  - 1977 Apple 1 & 2 (1MHz processor, 4-48KB memory, $1300-$2600)
  - 1979 Intel 8088
  - 1980 Seagate (then called Shugart) 5.25-in 5MB disk
  - 1981 IBM PC (4.77MHz, 16-640KB memory)
  - 1983 TCP/IP

- Computers were slow and expensive.


A history of C++ - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://portal.acm.org/citation.cfm?id=154766.155375&coll=ACM&dl=
Google
PORTAL
Purdue University Libraries
Subscribe (Full Service) Register (Limited Service, Free) Login
Search: The ACM Digital Library The Guide
SEARCH
THE ACM DIGITAL LIBRARY
Feedback
A history of C++: 1979–1991
Full text Pdf (3.04 MB)
Source History of Programming Languages archive
The second ACM SIGPLAN conference on History of programming languages table of
Cambridge, Massachusetts, United States
Pages: 271 - 297
Year of Publication: 1993
ISBN:0-89791-570-4
Also published in ...
Author Bjarne Stroustrup AT&T Bell Labs, Murray Hill, NJ
Sponsor SIGPLAN: ACM Special Interest Group on Programming Languages
Publisher ACM New York, NY, USA
Bibliometrics Downloads (6 Weeks): 24, Downloads (12 Months): 141, Citation Count: 8
Additional abstract references cited by index terms review collaborative colleagues
Done

C++

- design goals:
  - Simula's facilities for program organization
  - C's efficiency and flexibility
  - for system programming
- 1979-1983 C with Classes
- 1982-1985 C++
- 1985-1988 C++ 2.0
- 1988- standardization (ISO / ANSI)
- ISO = International Organization for Standardization
- ANSI = American National Standards Institute

Simula

- simulator for a distributed system
- class hierarchy
- capturing type errors by compiler
  - type: int, string, Student, Computer ...
  - type error, for example, a Student object + 3, a Computer object + "hello" ...
- problem of Simula: link time too long
  - run-time type checking
  - variable initialization
  - garbage collection, even for a program without garbage

  $\Rightarrow$ performance too low

Programming Language Design

- Never attack a problem with wrong tools.
- support for program organization: class, hierarchy, concurrency, static type checking
- good tools to compile files separately, to link files written in different languages, and to produce fast programs
- portable across different machines
- His background in OS and communication affects many design decisions, such as model of protection and exception handling
- **A good language requires a good implementation. Performance matters.**

C with Classes

- new language developed to analyze UNIX kernel: analyze network traffic and modularize kernel

$\Rightarrow$ develop an extension of C by adding tools

$\Rightarrow$ Some programming languages are developed for specific purposes and then are generalized.

- no primitives to express concurrency, use libraries instead (different from Java with built-in thread supports)
  - built-in support: consistent with language, but may cause unnecessary overhead to the users that do not need this feature
  - libraries: more flexibility but increase the overhead in system administration to ensure version compatibility
- C with Classes to be used anywhere C is used $\Rightarrow$ **efficiency** requirements eliminate built-in runtime safety checking

Features in C with Classes

- "philosophy": language designers should not force programmers to use a particular style; instead, designers should provide styles, practices, and tools to help programmers avoid well known traps ⇒ C allows low-level operations and type conversions, so does C with Classes
- features (1980): class, derived class, public / private access, constructor / destructor, call and return, friend class, type checking and conversion of function arguments
- features (1981): inline, default arguments, overloading of assignment operator
- C with Classes was implemented as pre-processor of C ⇒ portable across machines ⇒ **common approach** for language design today

C with Classes → pre-processor → C → C compiler → executable

Design Decisions in C with Classes

- A class is a type.
- Local variables are allocated at stack, not heap $\Rightarrow$ no need to call garbage collection
- Default access control is private.
- Static type checking for function arguments and return values.
- Class declarations and function definitions can be in different files (different from Java). Hence, class declaration can be the "interface" (Java distinguishes interface from class)
- "new" calls constructor (not all valid C programs are valid C++ programs)
- Use-defined types (classes) are treated in the same way as the built-in types.
- Function inlining is used to reduce the overhead of calls $\Rightarrow$ discourage programmers from declaring data members as public.

```cpp
// classX.h
class X {
public:
    void foo(int, float);
};
// both a class declaration and
// interface
```

```cpp
// classX.cpp
#include "classX.h”
void X::foo(int a, float b) {
    ...
}
// define the implementation of a
// member function
```

Garbage Collection in C++

- considered until 1985
- inappropriate for a language (C) already had run-time memory management
- GC would degrade performance unacceptably

- Stroustrup stressed that there was no "grand plan" to develop C++. Hence, the usefulness of the language resided on the ability to attract users in Bell Lab by solving their problems, efficiently.

1982 C++

- C with Classes was a "medium success"
- major features:
  - virtual function
  - function and operator overloading
  - reference
  - constant
- virtual function
  - to adapt to similar but different (common base class) types
  - a large if-then-else or switch-case block is undesirable
  - dilemma: allow adaptability by users without allowing the change of base classes (possibly from the library)

not object-oriented

```
void shape::draw()
{
    switch (type) {
    case circle:
            // draw a circle
            break;
    case square:
            // draw a square
            break;
    case triangle:
            // draw a triangle
            break;
    }
};
```

```
class Shape
{
    virtual void draw() = 0;
}
class Circle: public Shape
{
    void draw() ...// draw a circle
};

class Square: public Shape
{
    void draw() ... // draw a square
};
```

1986 C++ 2.0

- multiple inheritance, "the fundamental flaw in these arguments is that *they take multiple inheritance far too seriously*... it is quite cheap... you don't need it very often but when you do it is essential."
- type-safe linkage
- abstract class
- static member functions
- protected members
- overloading ->

- Exception handling was added later.

Summary

- C++ was developed to solve a specific problem: simulating distributed systems
- It is important to choose a good language as the base and build on top of the base; this can obtain immediate tool support.
- Features do not have to be added at once. Most features are added out of necessity, as the basic functionalities are available.
- Separate compilation and linking is critical for developing large-scale programs.
- Performance is essential. Many design decisions are based on the impact of performance.

Brief History of Java 1995-

- started in 1991 and announced in 1995
- Java started as a technology for entertainment "set-top box" to create a language that can run on small portable systems, not intended for system programming (as C++) ... but cable companies were unwilling to support
- The focus then switched to support Internet for processor (hardware) independent and operating-system independent (to be further discussed later)
- need: execute programs from remote machines through the Internet

$\Rightarrow$ A new language is more likely to succeed to solve a new problem. Solving an old problem is harder because of the existing programs and the infrastructures.

- 1994, a "better browser"

Success

- interactive browser:
  - With Java, users can interact with the browser, beyond browse, scroll, and click.
  - Sun Microsystem, as a primarily hardware company, developed Java to create the demand for high-performance networking equipment and computers.
  - 1995/03/23 San Jose Mercury News headline
  - Security is crucial since malicious code can easily propagate through the Internet (different goals from C++)
- widely used on
  - 4.5B devices
  - 1.5B phones
  - printer, webcam, game, car ...

Java: Sun vs. Microsoft

- 2002, Sun filed a lawsuit against Microsoft for violating the license agreement about Java
- Java was considered a threat to Microsoft's control of the operating system market.
- Sun accused that Microsoft modified Java in Windows and thus made it incompatible with other platform running Java.
- 2004, the two companies settled

- (background)
  - 1998 US antitrust against Microsoft, settled on 2001/11/02
  - 2003 European Union issued penalty to Microsoft
  - 2000-2002 Internet bubble burst
  - 2008/06/27 Bill Gate’s last day in Microsoft

	C++	Java
organization	AT&T Bell Lab	Sun Microsystem
target environment	system programming	embedded system Internet
base language	C	N/A
priority	efficiency	security
growth force	personal computer (to a lesser extent)	Internet
object-oriented	optional	mandatory
run-time array index checking	N/A	exception
memory management	destructor	garbage collection
global base class	N/A	Object
multiple inheritance	yes	interface

	C++	Java
concurrency	external library	built-in, thread
friend function / class	yes	N/A
parameter passing	value (primitive types), pointer, reference	value (primitive types), reference
virtual function	explicit	implicit
separate interface and implementation	yes (.h and .cpp)	N/A
exception handling	yes	yes
function overloading	yes	yes
default value of function parameters	yes	N/A
operator overloading	yes	N/A
graphics library	external	built-in AWT and SWING

Lessons Learned

- A successful language needs a clearly defined target. Creating a new language to replace an existing one is unlikely to succeed.
- Prioritize the requirements: efficiency for C++ and platform neutral for Java
- Tools (compiler, linker, debugger, runtime environment ...) and libraries (graphics, thread ...) are crucial, probably more important than the "elegance" of a language.
- Performance cannot be ignored. Any new language will be compared with C in terms of performance.
- Keep the non-essential portions of the new language the same as a popular existing language. Do not confuse users.
- Be aware of non-technical forces (such as legal issues)

ECE 462
Object-Oriented Programming using C++ and Java

Java Remote Method Invocation

Yung-Hsiang Lu
yunglu@purdue.edu

Rethink Function Call

Caller

```
...
pi = computePi(40);
...
```

Callee

```
LongFloat computePi(int numDigits)
{
    ....
    ....
}
```

The caller does not care how computePi obtain the result.

How to Implement the Callee?

- compute PI
- use a lookup table
- ask another machine to compute

```
LongFloat computePi(int numDigits)
{
    connect to a server
    send the request to the server
    wait for the response
    return the result to the caller
}
```

Remote Method Invocation

An Overview of RMI Applications (The Java™ Tutorials > RMI) - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

http://java.sun.com/docs/books/tutorial/rmi/overview.html

The Java™ Tutorials

Download the JDK
Search the Tutorials
Hide the TOC

« Previous • Trail • Next »   Home Page > RMI

An Overview of RMI Applications

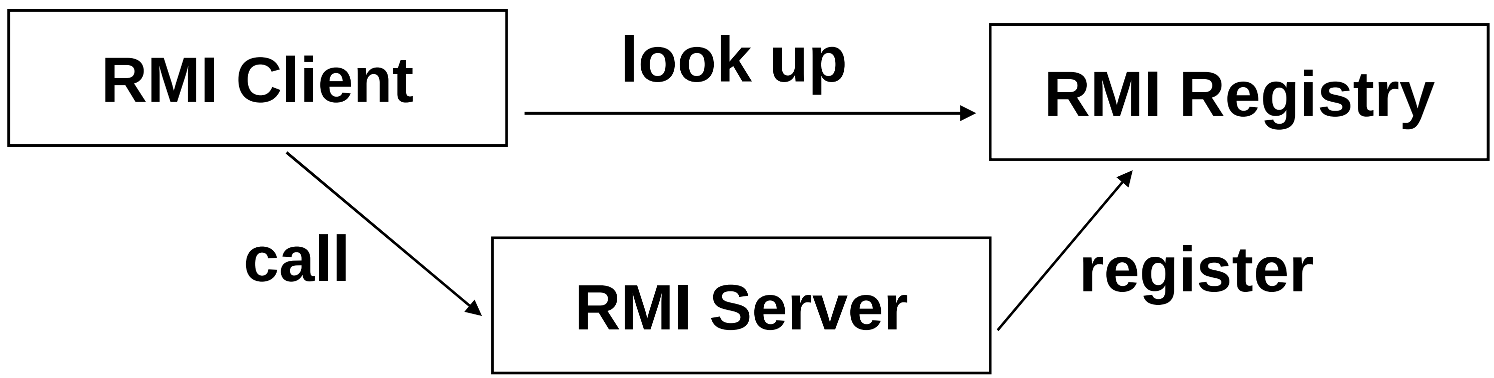
RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application*.

Distributed object applications need to do the following:

Done

RMI Architecture

- client-server model
- transmit objects to remote Java virtual machine



Compute Engine Example

Demonstration

1:qstruct05.ecn.purdue.edu - ee462b30@qstruct05 - SSH Secure Shell

File Edit View Window Help

```
[ECE 462 ] ls -R
.:
CVS/          RMIClient.policy  RMIServer.policy
Makefile      RMIInterface/
RMIClient/    RMIServer/

./CVS:
Entries  Repository  Root

./RMIClient:
ClientMain.java  ClientPI.java  CVS/

./RMIClient/CVS:
Entries  Repository  Root

./RMIInterface:
CVS/  EngineInterface.java  TaskInterface.java

./RMIInterface/CVS:
Entries  Repository  Root

./RMIServer:
CVS/  ServerEngine.java

./RMIServer/CVS:
```

qstruct04.ecn.purdue.edu - ee462b30@qstruct04 - SSH Secure Shell

File Edit View Window Help

```
[ECE462 ] javac RMIInterface/EngineInterface.java RMIInterf
ace/TaskInterface.java
[ECE462 ] jar cvf RMIInterface.jar RMIInterface/*class
added manifest
adding: RMIInterface/EngineInterface.class(in = 368) (out=
223)(deflated 39%)
adding: RMIInterface/TaskInterface.class(in = 248) (out= 16
5)(deflated 33%)
[ECE462 ] ls
RMIClient/  RMIInterface/  RMIInterface.jar  RMIServer/
[ECE462 ]
```

compile interface and create a jar file

qstruct04.ecn.purdue.edu - ee462b30@qstruct04 - SSH Secure Shell

File Edit View Window Help

```
[ECE462 ] javac -cp ./RMIInterface.jar RMIServer/ServerEngi
ne.java
[ECE462 ] ls RMIServer/
ServerEngine.class  ServerEngine.java
[ECE462 ]
```

build server

qstruct04.ecn.purdue.edu - ee462b30@qstruct04 - SSH Secure Shell

File Edit View Window Help

```
[ECE462 ] javac -cp ./RMIInterface.jar RMIClient/ClientMain
.java RMIClient/ClientPI.java
[ECE462 ] ls RMIClient/
ClientMain.class  ClientPI.class
ClientMain.java   ClientPI.java
[ECE462 ] 
```

build client

1:qstruct05.ecn.purdue.edu - ee462b30@qstruct05 - SSH Secure Shell

File Edit View Window Help

```
[ECE 462 ] rmiregistry &
[1] 10934
[ECE 462 ] java -cp /home/shay/a/ee462b30/lecturecode/1112/
rmi:/home/shay/a/ee462b30/lecturecode/1112/rmi/RMIInterface
.jar -Djava.rmi.server.codebase=file:/home/shay/a/ee462b30/
lecturecode/1112/rmi/RMIInterface.jar -Djava.rmi.server.hos
tname=qstruct05.ecn.purdue.edu -Djava.security.policy=RMISe
rver.policy RMIServer.ServerEngine
ServerEngine bound
```

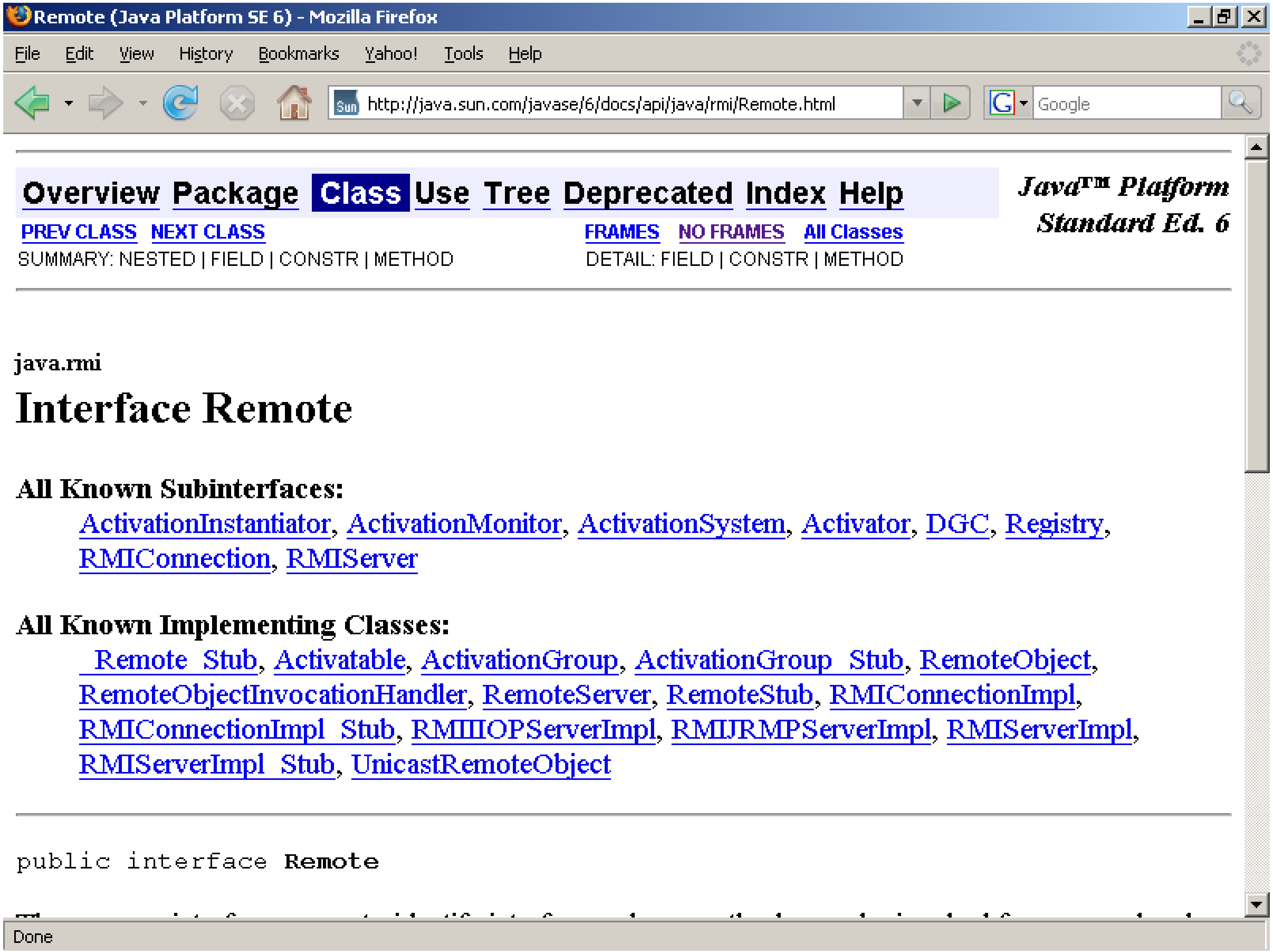
start server

3:qstruct05.ecn.purdue.edu - ee462b30@qstruct05 - SSH Secure Shell

File Edit View Window Help

```
[ECE 462 ] java -cp /home/shay/a/ee462b30/lecturecode/1112/
rmi:/home/shay/a/ee462b30/lecturecode/1112/rmi/RMIInterface
.jar -Djava.rmi.server.codebase=file:/home/shay/a/ee462b30/
lecturecode/1112/rmi/RMIInterface.jar -Djava.security.polic
y=RMIClient.policy RMIClient.ClientMain qstruct05.ecn.purdu
e.edu 65
3.141592653589793238462643383279502884197169399375105820974
94459231
[ECE 462 ]
```

execute client and print result



Overview Package **Class** Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 6

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.rmi

Interface Remote

All Known Subinterfaces:

ActivationInstantiator, ActivationMonitor, ActivationSystem, Activator, DGC, Registry, RMIConnection, RMIServer

All Known Implementing Classes:

_Remote_Stub, Activatable, ActivationGroup, ActivationGroup_Stub, RemoteObject, RemoteObjectInvocationHandler, RemoteServer, RemoteStub, RMIConnectionImpl, RMIConnectionImpl_Stub, RMIIIOPServerImpl, RMIJRMPServerImpl, RMIServerImpl, RMIServerImpl_Stub, UnicastRemoteObject

```
public interface Remote
```

Source Code

Name Mapping

	Sun's Tutorial	This Example
package	compute	RMIInterface
package	client	RMIClient
package	engine	RMIServer
interface	Compute	EngineInterface
interface	Task	TaskInterface
class	ComputeEngine implements Compute	ServerEngine implements EngineInterface
class	ComputePi	**Client**Main
class	Pi implements Task <BigDecimal>, Serializable	**Client**PI

File Edit Options Buffers Tools Java Help

```java
package RMIInterface;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface EngineInterface extends Remote {
    <T> T executeTask(TaskInterface<T> t)
        throws RemoteException;
}
```

--(Unix)-- EngineInterface.java (Java CVS:1.1.1.1 Abbrev)--L1--All-

File Edit Options Buffers Tools Java Help

```java
package RMIInterface;

public interface TaskInterface<T> {
    T executeCode();
}
```

--(Unix)--   TaskInterface.java   (Java CVS:1.1.1.1 Abbrev)--L1--All---

File Edit Options Buffers Tools Java Help

```java
package RMIServer;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import RMIInterface.EngineInterface;
import RMIInterface.TaskInterface;

public class ServerEngine implements EngineInterface {

    public ServerEngine() {
        super();
    }

    public <T> T executeTask(TaskInterface<T> t) {
        return t.executeCode();
    }

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "ECE462 Compute";
            EngineInterface engine = new ServerEngine();
```

--(Unix)-- ServerEngine.java   (Java CVS:1.1.1.1 Abbrev)--L1--Top----

File Edit Options Buffers Tools Java Help

```java
    }

    public <T> T executeTask(TaskInterface<T> t) {
        return t.executeCode();
    }

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "ECE462 Compute";
            EngineInterface engine = new ServerEngine();
            EngineInterface stub =
                (EngineInterface)
                UnicastRemoteObject.exportObject(engine, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("ServerEngine bound");
        } catch (Exception e) {
            System.err.println("ServerEngine exception:");
            e.printStackTrace();
        }
    }
}
```

--(Unix)--   ServerEngine.java   (Java CVS:1.1.1.1 Abbrev)--L21--Bot---

File Edit Options Buffers Tools Java Help

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.math.BigDecimal;
import java.net.InetAddress;
import RMIInterface.EngineInterface;

public class ClientMain {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "ECE462 Compute";
            Registry registry = LocateRegistry.getRegistry(args[0]);
            EngineInterface comp = (EngineInterface)
                registry.lookup(name);
            ClientPI task = new ClientPI(Integer.parseInt(args[1]));
            BigDecimal pi = comp.executeTask(task);
            System.out.println(pi);
        } catch (Exception e) {
            System.err.println("ClientMain exception:");
            e.printStackTrace();
        }
    }
}
```

--(Unix)-- ClientMain.java (Java CVS:1.1.1.1 Abbrev)--L18--Bot----

File Edit Options Buffers Tools Java Help

```java
package RMIClient;

import RMIInterface.TaskInterface;
import java.io.Serializable;
import java.math.BigDecimal;
import java.net.InetAddress;
public class ClientPI implements TaskInterface<BigDecimal>,
                                 Serializable {

    private static final long serialVersionUID = 227L;
    /** constants used in pi computation */
    private static final BigDecimal FOUR =
        BigDecimal.valueOf(4);
    /** rounding mode to use during pi computation */
    private static final int roundingMode =
        BigDecimal.ROUND_HALF_EVEN;
    /** digits of precision after the decimal point */
    private final int digits;
    /**
     * Construct a task to calculate pi to the specified
     * precision.
     */
    public ClientPI(int d) {
        digits = d;
    }
--(Unix)--  ClientPI.java      (Java CVS:1.1.1.1 Abbrev)--L7--Top-------
```

File Edit Options Buffers Tools Java Help

```java
    public BigDecimal executeCode() {
        return computePi(digits);
    }

    /**
     * Compute the value of pi to the specified number of
     * digits after the decimal point.  The value is
     * computed using Machin's formula:
     *
     *          pi/4 = 4*arctan(1/5) - arctan(1/239)
     *
     * and a power series expansion of arctan(x) to
     * sufficient precision.
     */
    public BigDecimal computePi(int digits) {
        int scale = digits + 5;
        BigDecimal arctan1_5 = arctan(5, scale);
        BigDecimal arctan1_239 = arctan(239, scale);
        BigDecimal pi = arctan1_5.multiply(FOUR).subtract(
                                  arctan1_239).multiply(FOUR);
        return pi.setScale(digits,
                           BigDecimal.ROUND_HALF_UP);
    }
    /**
     * Compute the value, in radians, of the arctangent of
```

--(Unix)-- ClientPI.java   (Java CVS:1.1.1.1 Abbrev)--L49--28%------

File Edit Options Buffers Tools Java Help

```
    public static BigDecimal arctan(int inverseX,
                                    int scale)
    {
        BigDecimal result, numer, term;
        BigDecimal invX = BigDecimal.valueOf(inverseX);
        BigDecimal invX2 =
            BigDecimal.valueOf(inverseX * inverseX);
        numer = BigDecimal.ONE.divide(invX,
                                      scale, roundingMode);
        result = numer;
        int i = 1;
        do {
            numer =
                numer.divide(invX2, scale, roundingMode);
            int denom = 2 * i + 1;
            term =
                numer.divide(BigDecimal.valueOf(denom),
                             scale, roundingMode);
            if ((i % 2) != 0) {
                result = result.subtract(term);
            } else {
                result = result.add(term);
            }
            i++;
        } while (term.compareTo(BigDecimal.ZERO) != 0);
--(Unix)--  ClientPI.java      (Java CVS:1.1.1.1 Abbrev)--L71--68%------
```

Frequently Asked Questions - RMI and Object Serialization - Mozilla Firefox

http://java.sun.com/j2se/1.4.2/docs/guide/rmi/faq.html#whyserial

Object Serialization

1. Why must classes implement `Serializable` in order to be written to an `ObjectOutputStream`?

The decision to require that classes implement the `java.io.Serializable` interface was not made lightly. The design called for a balance between the needs of developers and the needs of the system to be able to provide a predictable and safe mechanism. The most difficult design constraint to satisfy was the safety and security of classes for the Java programming language.

If classes were to be marked as being serializable the design team worried that a developer, either out of forgetfulness, laziness, or ignorance might not declare a class as being `Serializable` and then make that class useless for RMI or for purposes of persistence. We worried that the requirement would place on a developer the burden of knowing how a class was to be used by others in the future, an essentially unknowable condition. Indeed, our preliminary design, as reflected in the alpha API, concluded that the default case for a class ought to be that the objects in the class be serializable. We changed our design only after considerations of security and correctness convinced us that the default had to be that an object not be serialized.

Security restrictions

Done

Makefile<2>

File Edit Options Buffers Tools Makefile Help

```
INT = RMIInterface
SER = RMIServer
CLI = RMIClient

build:
	javac $(INT)/EngineInterface.java $(INT)/TaskInterface.java
	jar cvf RMIInterface.jar $(INT)/*class
	javac -cp ./RMIInterface.jar $(SER)/ServerEngine.java
	javac -cp ./RMIInterface.jar $(CLI)/ClientMain.java $(CLI)/ClientPI.jav

clean:
	rm -f RMIInterface.jar
	rm -f $(INT)/*.class
	rm -f $(SER)/*.class
	rm -f $(CLI)/*.class

start:
	rmiregistry &
	java -cp /home/shay/a/ee462b30/lecturecode/1112/rmi:/home/shay/a/ee462b
12/rmi/RMIInterface.jar -Djava.rmi.server.codebase=file:/home/shay/a/ee462b30/l
mi/RMIInterface.jar -Djava.rmi.server.hostname=qstruct05.ecn.purdue.edu -Djava.
MIServer.policy RMIServer.ServerEngine

test:
	java -cp /home/shay/a/ee462b30/lecturecode/1112/rmi:/home/shay/a/ee462b
12/rmi/RMIInterface.jar -Djava.rmi.server.codebase=file:/home/shay/a/ee462b30/l
mi/RMIInterface.jar -Djava.security.policy=RMIClient.policy RMIClient.ClientMai
urdue.edu 65
```

RMIServer.policy

File Edit Options Buffers Tools Help

```
grant codeBase "file:/home/shay/a/ee462b30/lecturecode/1112/rmi" {
    permission java.security.AllPermission;
};
```

```
--(Unix)--  RMIServer.policy      (Fundamental CVS:1.1.1.1)--L5--All----
```

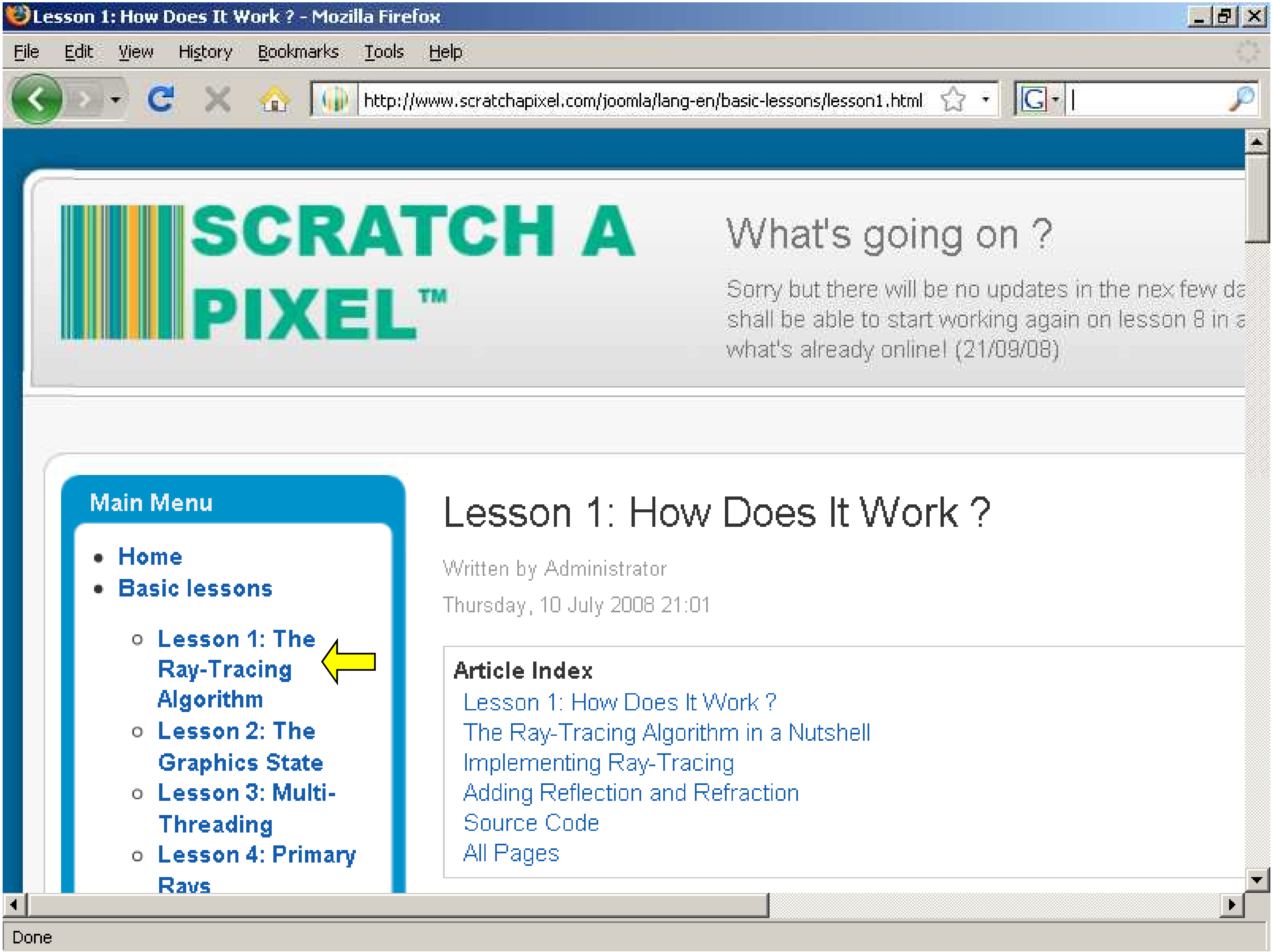
```
grant codeBase "file:/home/shay/a/ee462b30/lecturecode/1112/rmi" {
    permission java.security.AllPermission;
};
```

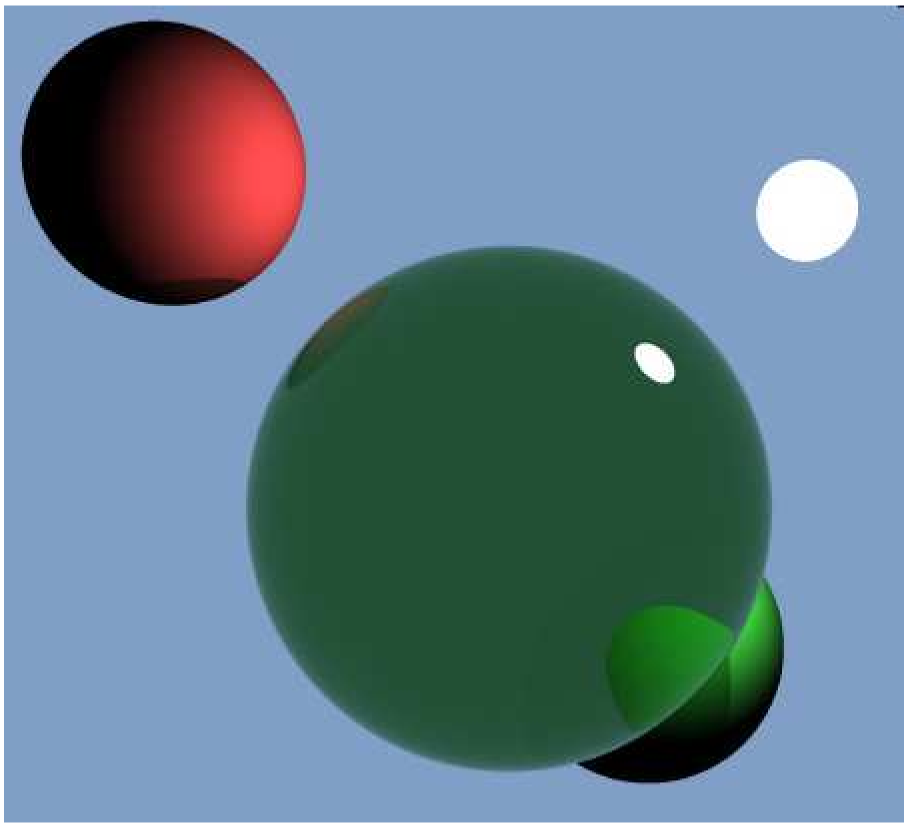
```
--(Unix)--  RMIClient.policy      (Fundamental CVS:1.1.1.1)--L1--All----
```

ECE 462
Object-Oriented Programming using C++ and Java

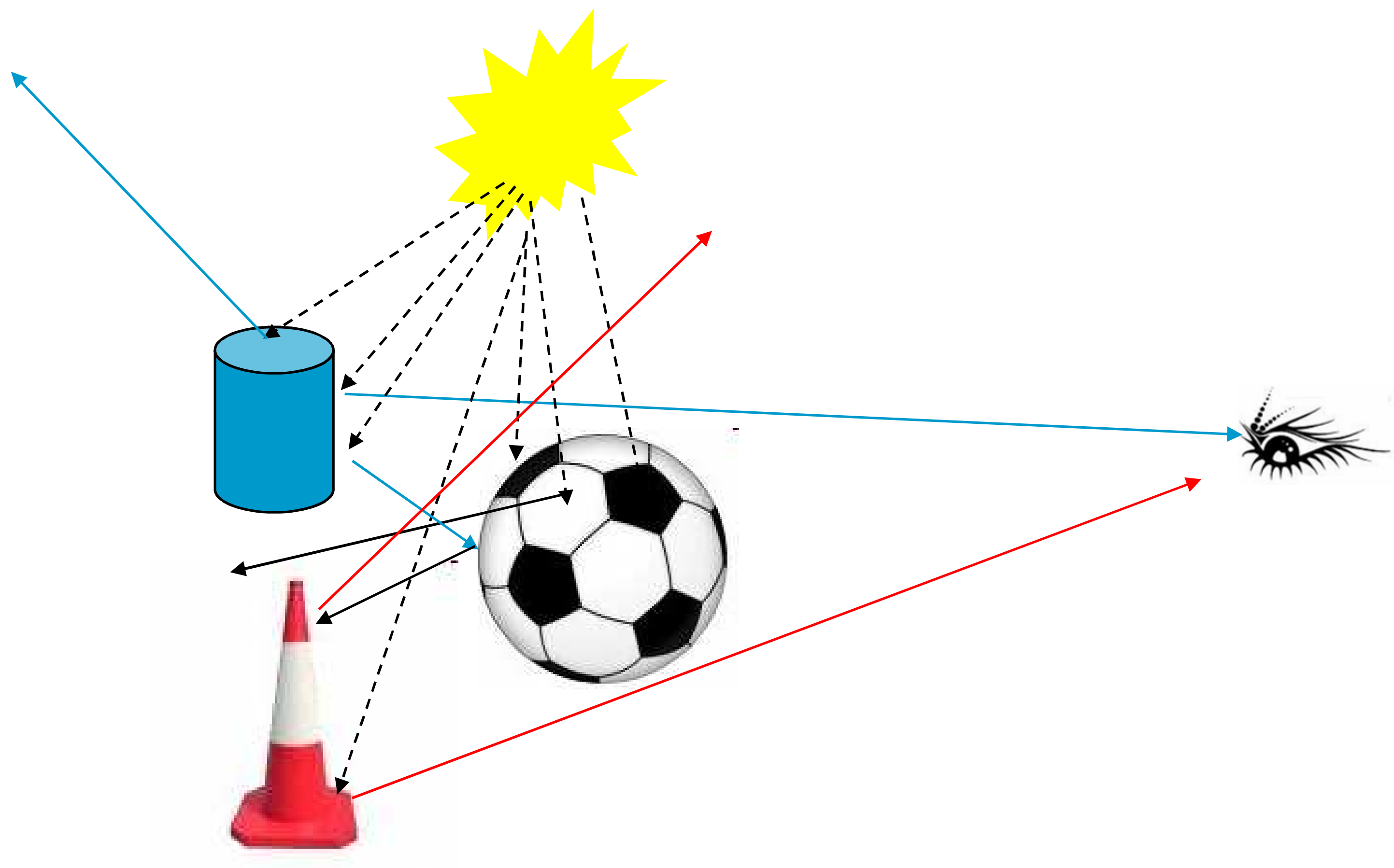
Ray Tracing

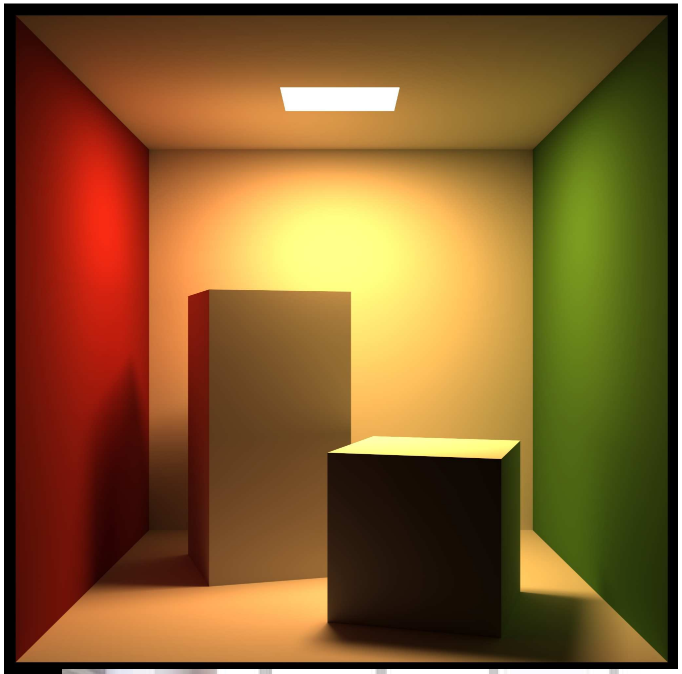
Yung-Hsiang Lu
yunglu@purdue.edu


Lesson 1: How Does It Work ? - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.scratchapixel.com/joomla/lang-en/basic-lessons/lesson1.html
SCRATCH A PIXEL™
What's going on ?
Sorry but there will be no updates in the nex few da
shall be able to start working again on lesson 8 in a
what's already online! (21/09/08)
Main Menu
Home
Basic lessons
Lesson 1: The Ray-Tracing Algorithm
Lesson 2: The Graphics State
Lesson 3: Multi-Threading
Lesson 4: Primary Rays
Lesson 1: How Does It Work ?
Written by Administrator
Thursday, 10 July 2008 21:01
Article Index
Lesson 1: How Does It Work ?
The Ray-Tracing Algorithm in a Nutshell
Implementing Ray-Tracing
Adding Reflection and Refraction
Source Code
All Pages
Done

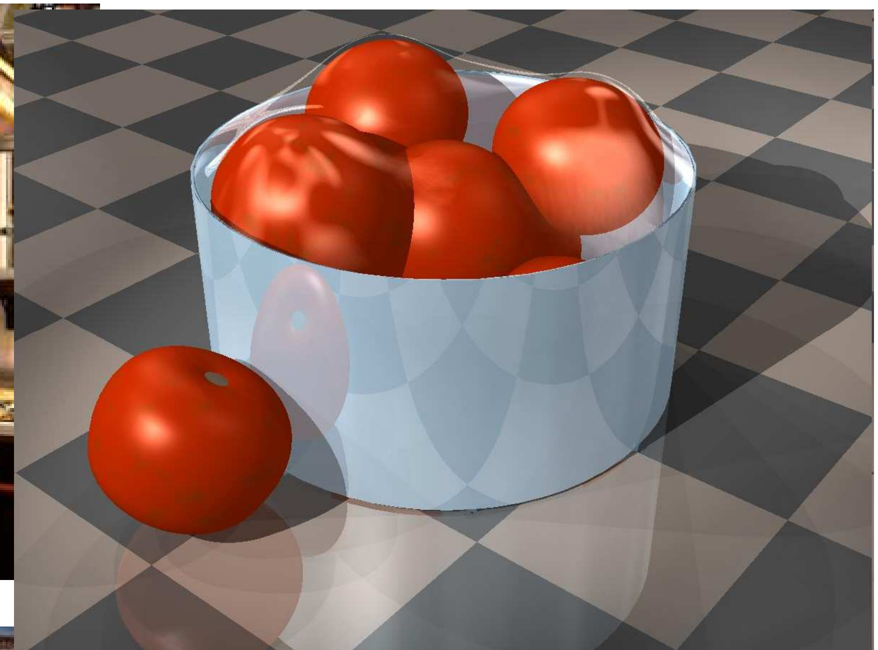


Ray Tracing





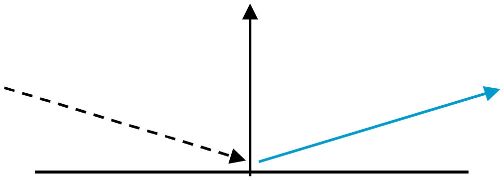

Rasterized
Ray traced

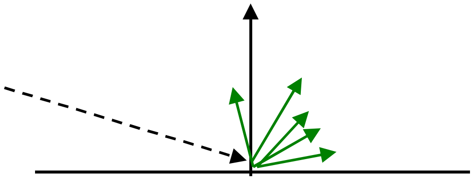



ROUTE
66
Rust-eze

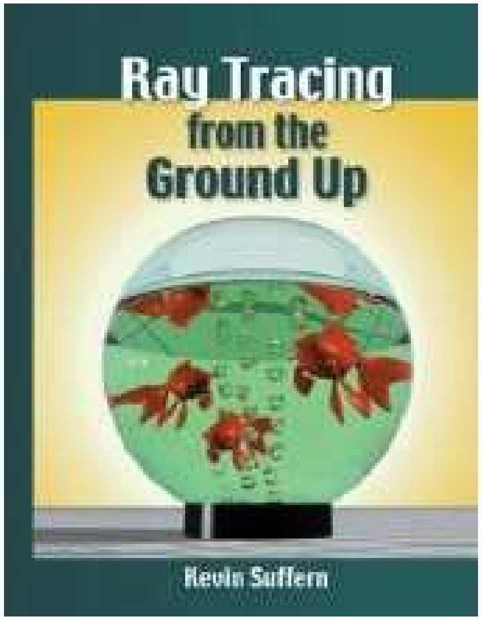
Advantages

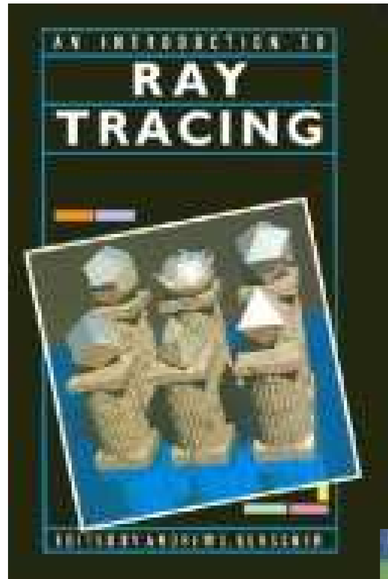
- generate high quality images:
  - model different light sources
  - surface properties

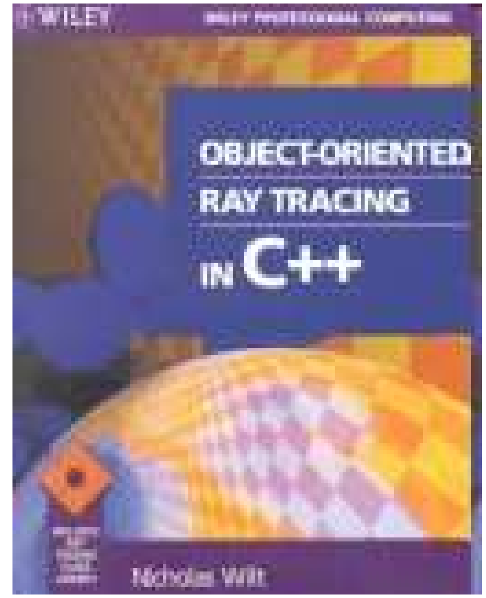


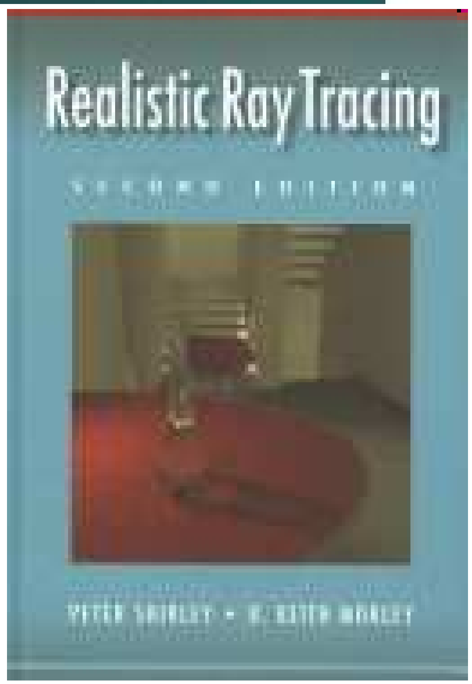


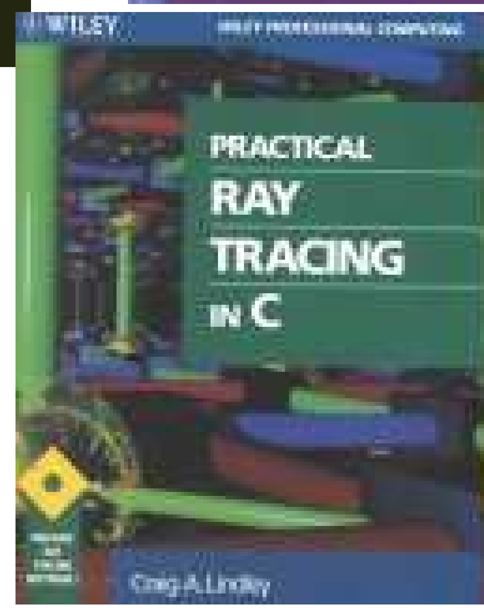
Books

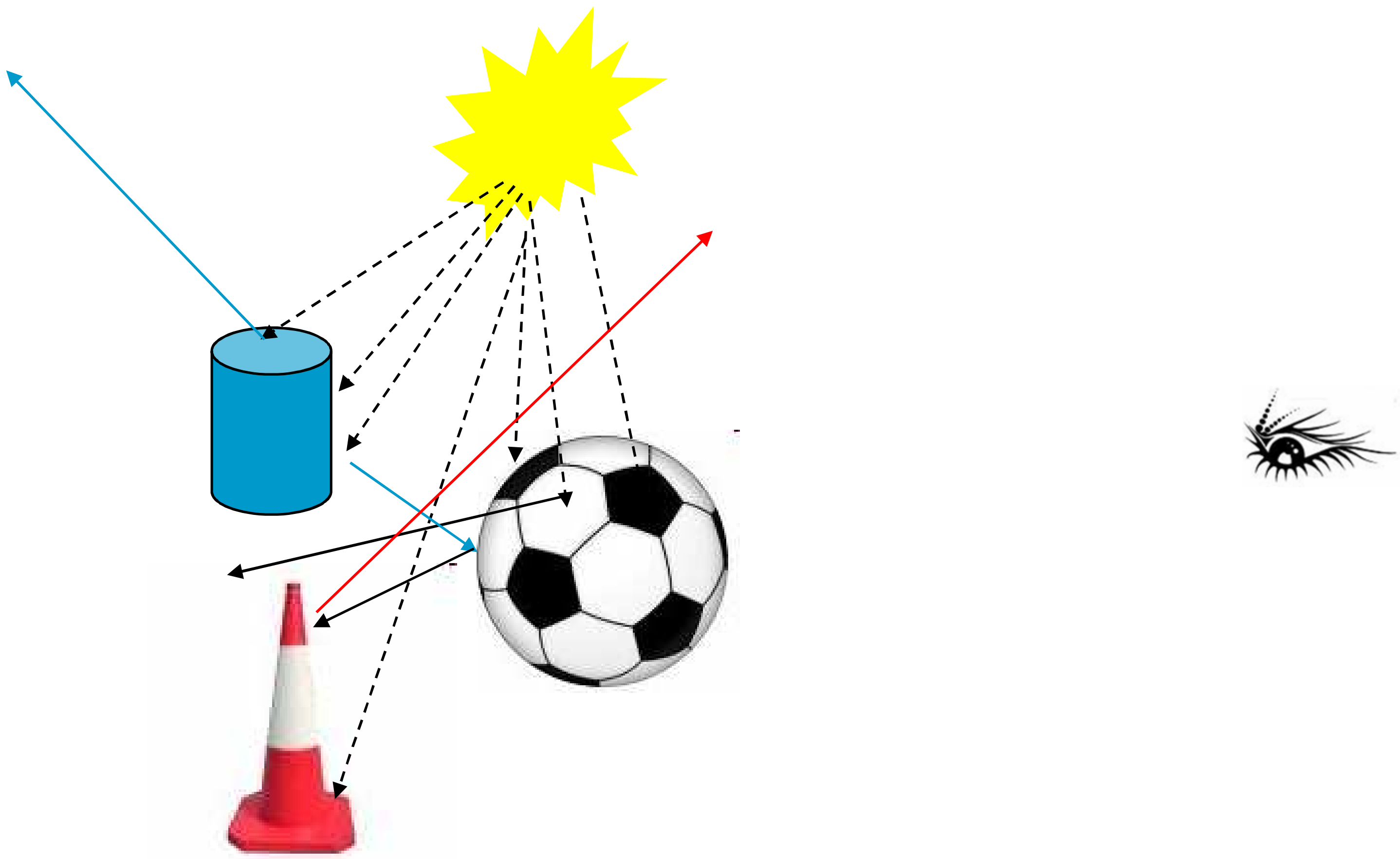

Ray Tracing
from the
Ground Up
Kevin Suffern


AN INTRODUCTION TO
RAY
TRACING

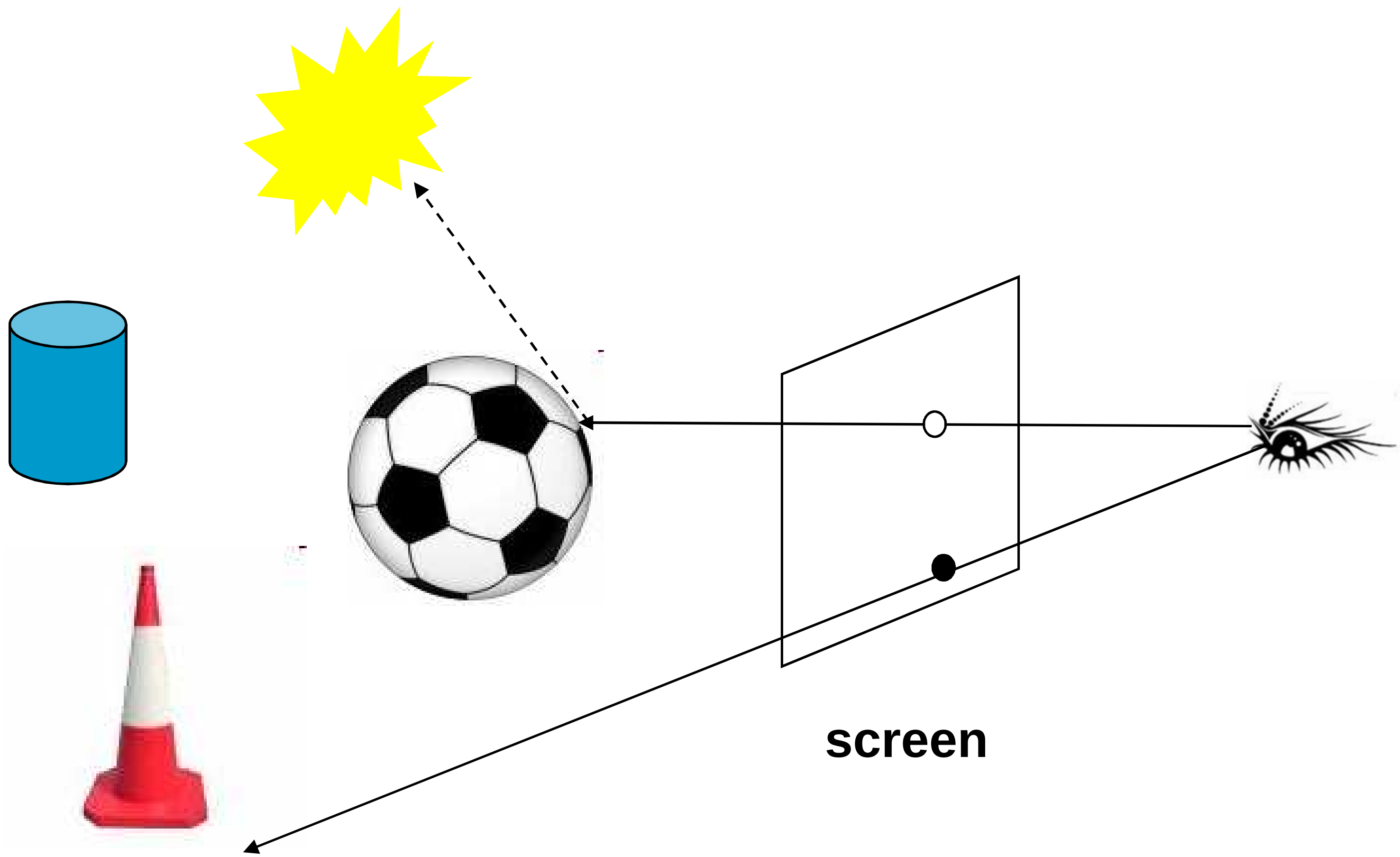

WILEY
OBJECT-ORIENTED
RAY TRACING
IN C++
Nicholas Wilt


Realistic Ray Tracing
SECOND EDITION
PETER SHIRLEY • R. KEITH MORLEY


WILEY
PRACTICAL
RAY
TRACING
IN C
Craig A. Lindley



Reverse Ray Tracing



Refraction and Ray Tracing

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{\lambda_1}{\lambda_2}$$