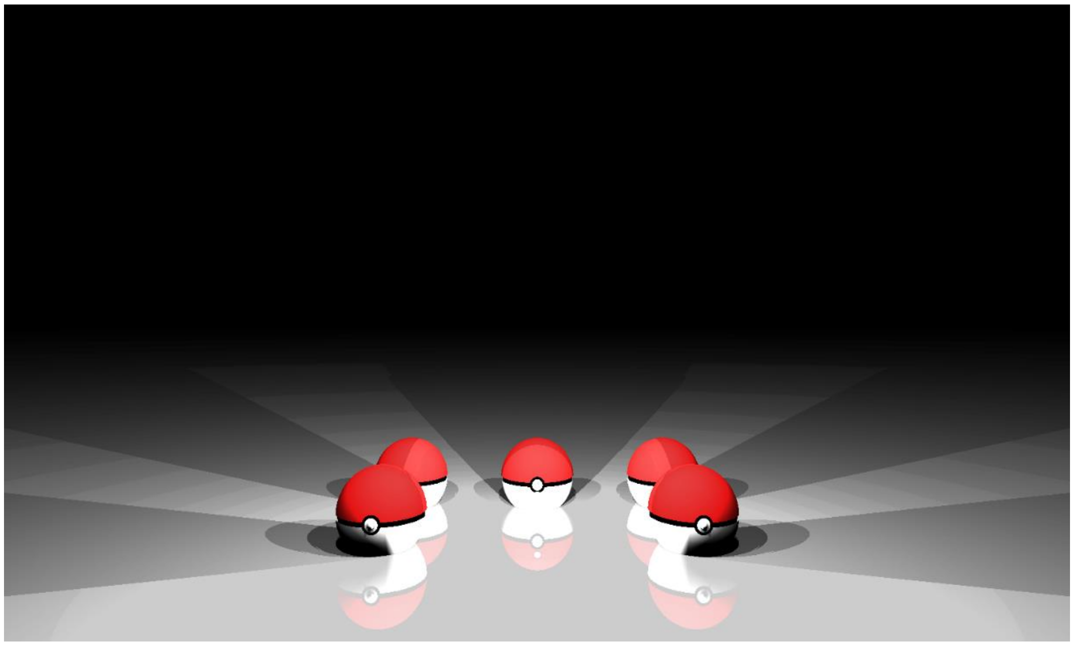
RAYTRACER



02-Feb-14

Raytracer project for Ahlia University

Raytracer

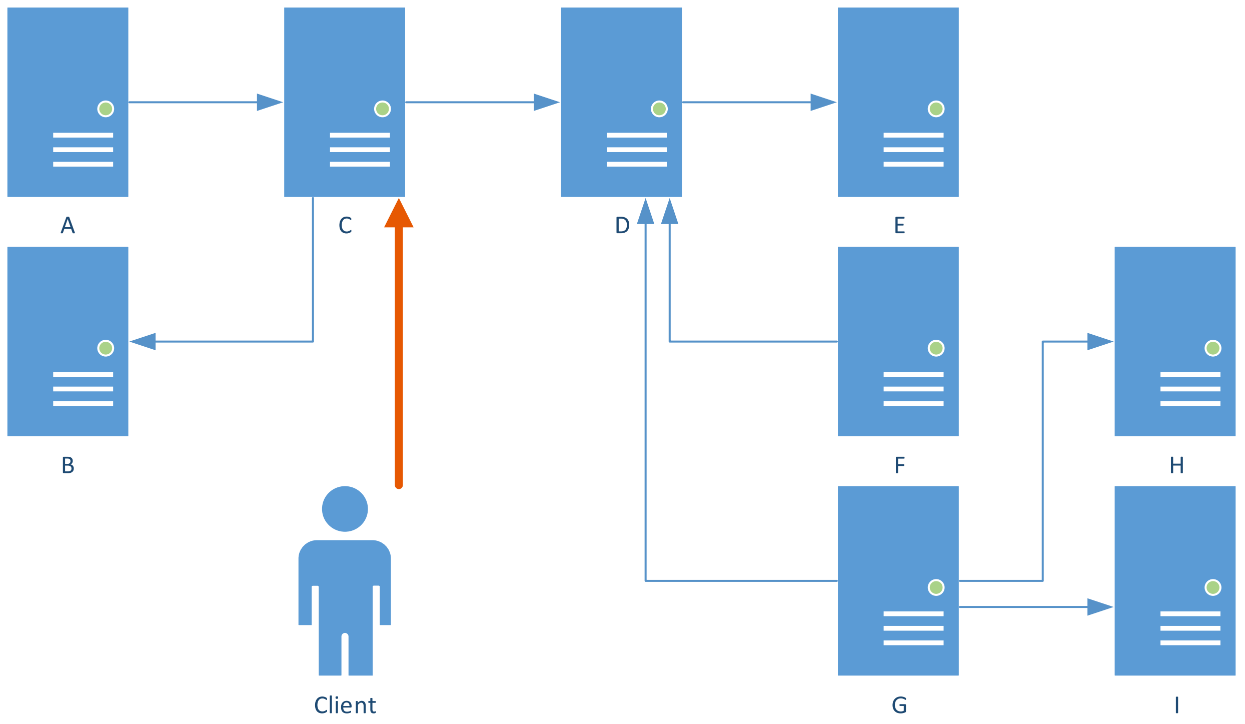
RAYTRACER PROJECT FOR AHLIA UNIVERSITY

Intro

In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects. The technique is capable of producing a very high degree of visual realism, usually higher than that of typical scanline rendering methods, but at a greater computational cost. This makes ray tracing best suited for applications where the image can be rendered slowly ahead of time, such as in still images and film and television visual effects, and more poorly suited for real-time applications like video games where speed is critical. Ray tracing is capable of simulating a wide variety of optical effects, such as reflection and refraction, scattering, and dispersion phenomena. (Wikipedia)

The main problem of this generation technic is the slow time. We will use the learning obtained from Distributed System session to improve the generation time using multiple servers and one client.

COMBES Valentin
MONSSARAT Bastien
BARRE-VILLENEUVE Thibault



The server will be linked using TCP sockets and a binary protocol *(cf network protocol)*.

In order to generate easily a network map, we may use multiple virtual machines using Linux. One configuration file per server will describe the neighbor server.

FOR EXAMPLE, THE CONFIGURATION FILE FOR THE "G" SERVER WILL DESCRIBE CONNECTION WITH THE "D", "H" AND "I" SERVER.

The client will be able to connect to any of the server, and this server will be considered as the master server (will distribute the work to the other one).

Multiple clients may be connected at the same time on one or different server.

The client will be a C# application for Windows.

Network protocol

The network protocol will be based on a TCP socket on port 11424 (randomly based on the last Pizza Hut due). The protocol will have a few instructions:

Handshake

0.	SAUTH	[ID / -1]	
1.	CAUTH	[ID / -1]	
2.	WELCOME	id	
3.	IDCH	oldId	[newId / -1]
4.	<unused>		
5.	CONFIRM	id	

Disconnect messages

6.	QUIT	src
7.	NETSPLIT	src

Job manager

8.	NEWJOB	src	size	content
9.	ENDJOB	id		
10.	READY	src		
11.	<unused>			

Computing

12.	CALC	job	dst	X	Y
13.	RESULT	job	X	Y	COLOR
14.	COMPILFAIL	job	src		
15.	MONITOR	#nbProc, ramLvl, proc1, proc2, proc3			
16.	CLIENTMONITOR	(char)cpu, (uint) usedRam, (uint) avalaibleRam			

The size of all id will be 2B (unsigned short), the coordinates of a plot will be 2B (unsigned short), the colors 4B (ARGB unsigned int). File size will be 4B (unsigned int, max size: 4GB).

Handshake

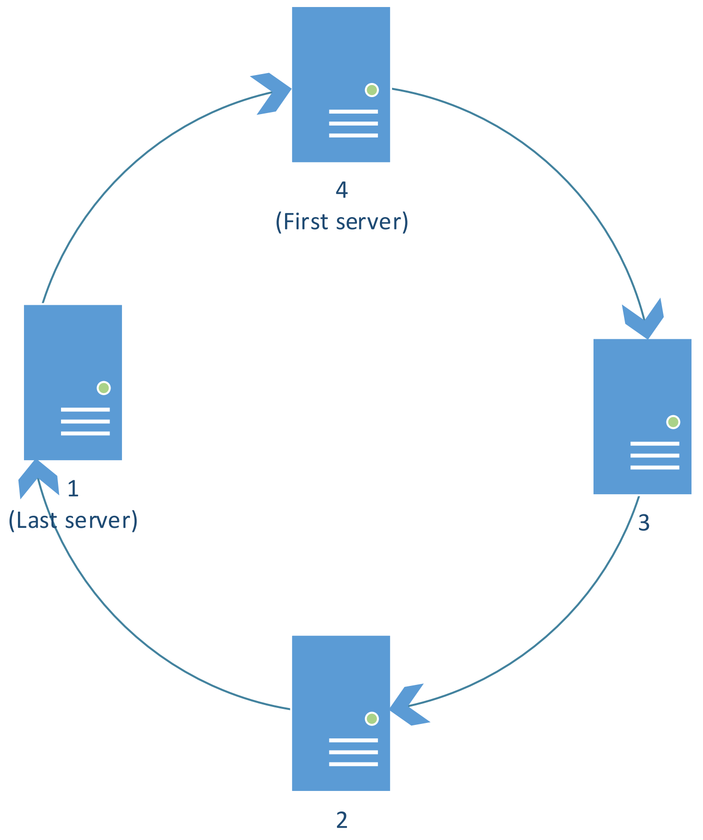
AUTHENTICATION: SERVER TO SERVER

- The *SAUTH* message will be sent from connecting server to listening server at each connection. The receiving server will reply WELCOME *id* to the first one, so that the server know his id. If the connecting server has server connected to him, he MUST send an *IDCH* message to inform them of his new id, wait for a *CONFIRM id packet*, then sending them a *RELOG* message.
- If a server receive a *RELOG packet*, he MUST request a new id to the newly connected server. To do that, he will send an *IDCH oldId -1* to his parent server BEFORE the packet propagation. The root server will answer an *IDCH oldId newId* packet. The server can now send the *RELOG* packet to its children.
- When a server have to send an *IDCH* package, it will send an *SAUTH* or a *IDCH* packet to its children (except the target child)
- If a server receive a WELCOME AND a SAUTH with the same id, we have a circle-loop problem. The connection MUST be aborted (close socket with no message)

AUTHENTICATION: CLIENT TO SERVER

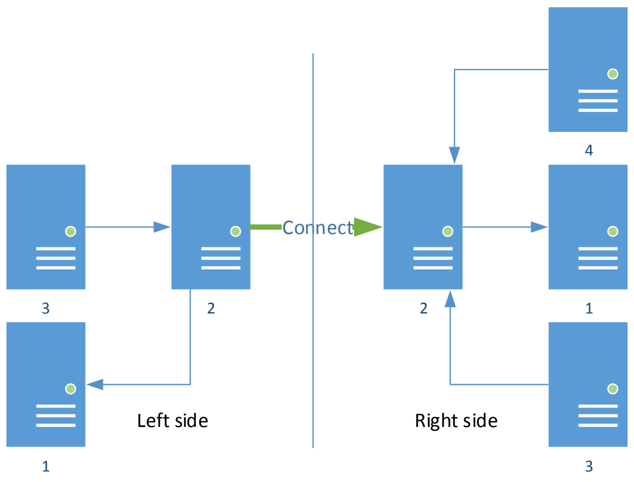
- The client will send a *CAUTH -1* message to identity himself. The server will reply *WELCOME id* and send *CAUTH id* to all of its children.

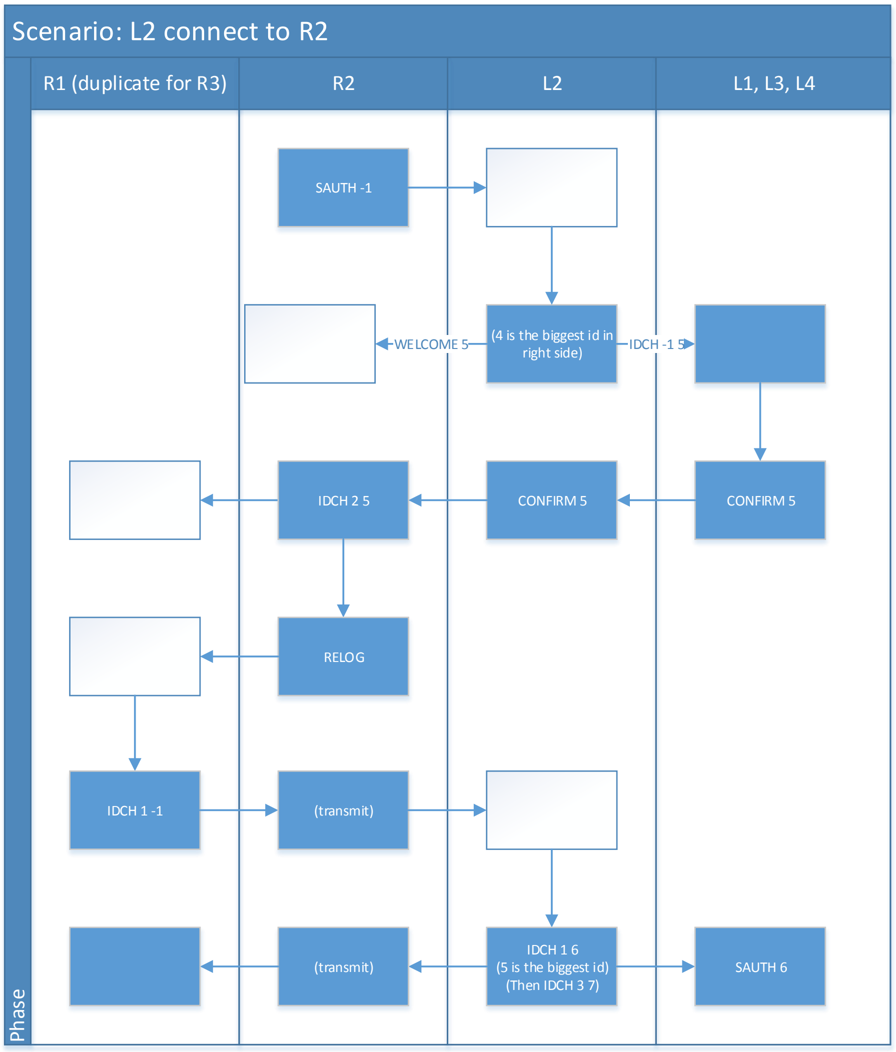
SCENARIO EXAMPLE 1: CIRCLE-LOOP PROBLEM



SRC	DST	MESSAGE
1	4	SAUTH
4	1	WELCOME 5
4	3	IDCH 5 -1
3	2	IDCH 5 -1
2	1	IDCH 5 -1

SCENARIO EXAMPLE 2: SERVER TO SERVER HANDSHAKE





Distributed system

Ahlia University

For every next message, the “repeat” behavior will be employed: the server send the received information to every children, avoiding to send back a packet.

Exception: The *CALC* and *RESULT* messages will be sent only towards the destination.

Disconnect messages

There is two possibilities for disconnecting a client:

- Technical issue – involuntary disconnect
- Voluntary shutdown

In both case, the server that has been isolated will send to everyone a *QUIT id* OR *NETSPLIT id* if the disconnect is clean or not.

Note: A *QUIT* or *NETSPLIT* on a client will terminate his project.

Job manager

Because of the possibility of multi-client, the server MUST be able to store the source code of multiple projects.

To distinguish between the different projects, we will use the id of the client.

NEWJOB

The NEWJOB command permit to setup a new project. The parameters are:

- Src: the id of the client
- Size: the size of the binary-encoded scene to render
- Content: The binary-encoded scene

ENDJOB

The *ENDJOB* will terminate the given job.

Parameter: id

READY

The ready command indicate that the server is ready for next instruction. This can occur when the server use multi-threads or when the server has finished all his instructions.

Computing

There will be two main commands to do the calculus: *CALC* and *RESULT*.

CALC will ask the server to compute the pixel color at the given coordinates.

Parameters:

- Project id: the id of the project, to know which configuration file use
- Destination id: Id of the server selected for computing
- X and Y: The coordinates of the point to compute.

RESULT permit to report the result of the operation. The parameters are:

- Project id: the user to report
- X and Y: The computed coordinates
- Color: the pixel color

The last command, *COMPFAIL*, is sent if the server is unable to process the project. In this case, the master server will treat it as inaccessible for the project.

Server directories

The server's files will be dispatched into these main directories and files:

• Src	Source directory
o Kernel	Main engine. Contains a Makefile and sources
o Modules	Modules source directory
▪ Objects	Objects module source.
▪ Shaders	contains sources and Makefile
▪ Effects	(One directory per module)
• Modules	Modules directory
o Objects	Objects modules directory (ex: sphere.lib)
o Shaders	Shaders modules directory
o Effects	Effects modules directory (ex: rotate.lib)
• Makefile	Compilation configuration file
• Rt	binary
• Server.conf	Configuration file for server

We can imagine these as a shared folder for an easy deployment on each virtual machine.

Ray tracing algorithm

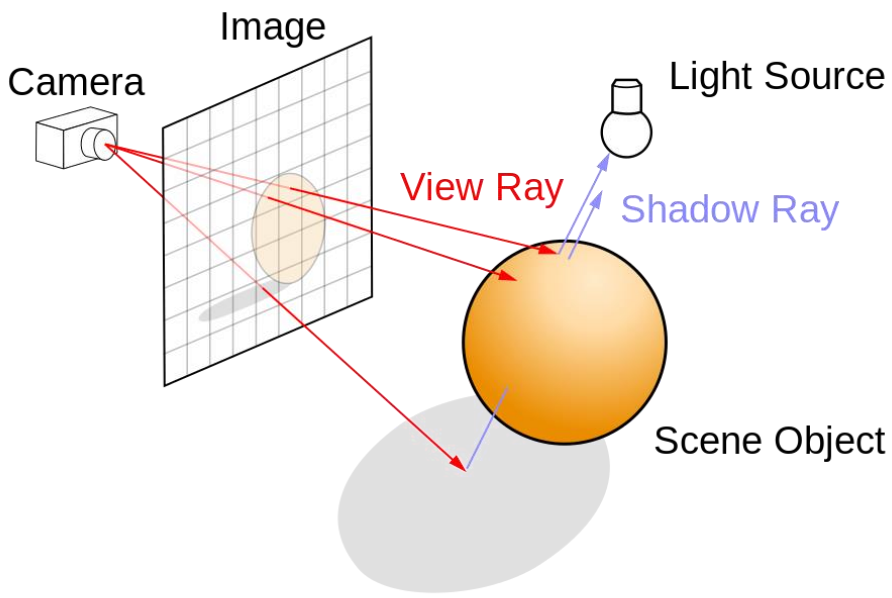
The ray tracing method is used to generate images. The principle is simple:

- We send a “solar ray” from the eye through each pixel of the view screen. For example, if we want to generate a 1600x900 image, we will have to send at least 1600 * 900 = 1’440’000 rays)
- For each ray, we will check if the ray touch one of the object, using mathematical formulas. If we don’t, we can return BLACK, or BACKGROUND.
- Else, we get the closest object, and return its color.

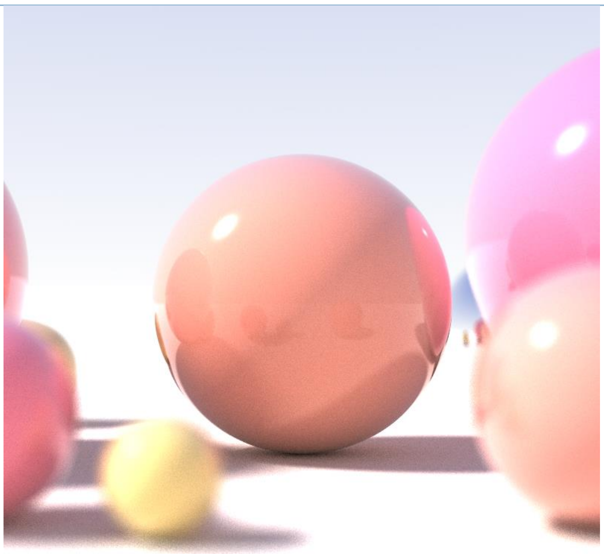
This is the beginning. Because there is different mathematical formulas for each object type, we will use modules for each object (see /modules/object)

We can now add effects such as shaders, reflexion, transparency, shadow and lighting (for example) by computing the impact of the ray on the object. (Will the ray be stopped? where are the nearby luminous sources? Will the ray continue somewhere else and what is the new vector?)

So a new ray (multiple?) may be sent from the first “impact” and the algorithm will start again from step 2.



Samples



Module management

At the launch time, the server will execute a *READDIR* on the /modules/ [objects, shaders, Effects] directories, will load all the shared libraries. Each library will implement one function called loadLibrary which will return an interface according the library type (object, shaders or effect).

These interfaces will be decided during development.

Configuration files

The server configuration file will be Linux-like, e.g. the following syntax:

```
[Section name]
Variable name = value
```

The scenes files to be passed throw the client application will be decided later, according to the needs during the development.

Compiled scenes

The scenes will be "compiled" by the client in order to optimize the transaction between the servers. The scene will be encrypted in binary; the structures are not decided yet and are up to be changed during the project.

Versioning

To permit the integration and the teamwork, we will be using Git on an Archlinux (Linux) server (throw Ssh connection).