|  |  |
| --- | --- |
|  | Ray tracer |
|  |  |
| 28-Apr-14 | Ray tracing project for Ahlia University (Distributed System) |
|  |  |

Ray tracer

Ray tracing project for Ahlia University (Distributed System)

# 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 use the learning obtained from Distributed System session and Operating System to improve the generation time using multiple servers and one client by splitting the calculus between each thread of all distributed servers.

# Network architecture



The server are linked using TCP sockets and a binary protocol *(see* [*network protocol*](#Net_proto)*)*.

In order to generate easily a network map, multiple server with GNU/Linux are used. We also use some virtual machine to simulate a bigger network. Every server have a configuration file describing the neighboring server.

For example, the configuration file for the “G” server describes connection with the “D”, “H” and “I” server.

The client is able to connect to any of the server from the moment the “No Listen” instruction is NOT specified in the configuration file. The server connected to the client is considered as the super-peer for the client’s image, and is used to distribute the work to the other peers.

Multiple clients can be connected at the same time on one or different server, and they all have a different id, which also represent a project. The client can only hold one project.

The client will be a C# application for Windows.

# Network protocol

The network protocol will be based on a TCP socket. The protocol will have these instructions:

## Handshake

### SAUTH

Parameter(s): [ID / -1] [Sub server number]

If ID is -1, these allow a new client to register as a server. The target server will answer a WELCOME message and broadcast SAUTH ID NBSERVER.

If ID is different than -1, it is used to inform servers of the presence of a new one with the ID. The target server will answer a CONFIRM message, or disconnect if ID match its own ID.

### CAUTH

Parameter(s): [ID / -1]

If ID is -1, this packet permit a new client to register as a CLIENT. This initiate a new ID, and the server will reply WELCOME, and broadcast CAUTH ID.

If ID if not -1, this packet is used to inform the server of a new client.

### WELCOME

Parameter(s): id

This packet permit to inform a client or a server of is ID. In the case of a server, the receiving server will have to wait for a CONFIRM message to be considered as a part of the network.

### IDCH

Parameter: old\_Id [new\_Id / -1]

This permit to inform the network of an ID change in the network. The OLDID id has to be replaced by NEWID.

If old\_Id equals 0xFF (255), this packet inform the receiving server of the sending server’s ID (see CONFIRM)

### CONFIRM

Parameter: ID

If the ID is the receiving server’s ID, it means that the server has been confirmed as a part of the network.

If the ID is not the receiving server’s ID, it means that the server is considered as a part of the network with no issues in the server and his sub-network.

## Disconnect messages

### QUIT

Parameter: ID

This inform the network of a client disconnect, with no problem. The server has to be released.

### NETSPLIT

The same as QUIT, but the server close the connection in a bad way (crash, shutdown, ping timeout).

(See QUIT)

## Job manager

### NEWJOB

Parameter(s): Client\_Id Binary\_size, content

This start the calculus of a new scene. If the client is connected direct to the server, the server has to manage the scheduling of the scene.

Client\_id is considered as the scene’s ID, as one client can only manage one scene.

Binary\_Size, Binary\_content: The binary scene. This is a compiled version of the scene (see Binary scene protocol)

### ENDJOB

Parameter(s): id

This stop the calculus of the scene. Can abort a running operation.

### READY

Parameter(s): ID

This inform the network that the server ID finished all its tasks and is ready to perform a new one.

## Computing

### CALC (Deprecated)

Parameter(s): job destination X Y

This command ask the server DESTINATION to compute and return as RESULT the pixel color located at [X, Y] for the given ID job.

### RESULT (Deprecated)

Parameter(s): job X Y COLOR

This return the pixel color of the JOB at position [X, Y]

### CHUNK\_CALC

Parameter(s): job destination X Y Width Height

This command ask the server DESTINATION to compute and return as CHUNK\_RESULT the pixels color located in the rectangle at [X, Y] and of size Width/Height for the given ID job

### CHUNK\_RESULT

Parameter(s): job X Y Width Height COLOR, Color, Color, Color…

This return the pixels colors of JOB in the rectangle located at [X, Y] and of size Width/Height.

The following arguments are an ensemble of (Width x Height) color

### COMPILFAIL

Parameter(s): job src

This permit to exclude SRC from the calculus of the scene. Mainly cause by the lack of a module, it can leads to the impossibility of computing the scene.

If the client receive this packet, it means that no server can handle the scene.

### MONITOR

Parameter(s): #nbProc, ramUsed, ramMax, swapLvl, swapMax, proc1, proc2, proc3

This inform all the network of the statistics of all servers. It hold the number of processors, the percentage of utilization of every processors, the RAM level and the SWAP (Interchange Linux space) utilization.

The processor utilization is sent as *percentage \* 255* in one Byte, for optimization.

### CLIENTMONITOR (char)cpu, (uint) usedRam, (uint) avalaibleRam

This packet is only for client, to inform them about the global use of the resources of the network. The differences with MONITOR packet is the CPU (average CPU is sent), and the SWAP and RAM are summed.

## Unit management

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).

## Scenario example: Assigning Unique Identifier



|  |  |  |
| --- | --- | --- |
| Source | Destination | message |
| D | B | SAUTH -1 0 |
| B | D | WELCOME 4 |
| B | A | SAUTH 4 0 |
| B | C | SAUTH 4 0 |
| A | B | CONFIRM 4 |
| C | B | CONFIRM 4 |
| B | D | CONFIRM 4 |
| D | B | IDCH 0 1 |
| D | B | SAUTH 2 |
| D | B | SAUTH 3 |

# Server directories

The server’s files are dispatched into these main directories and files:

* Src Source directory
  + Kernel Main engine. Contains a Makefile and sources
  + Modules Modules source directory
    - Objects, Shader, Effects

Objects module directories. Contains Makefiles and sources

* Modules Modules directory
  + Lib.so Library files
* Makefile Compilation configuration file
* drt binary
* Server.conf Configuration file for server

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

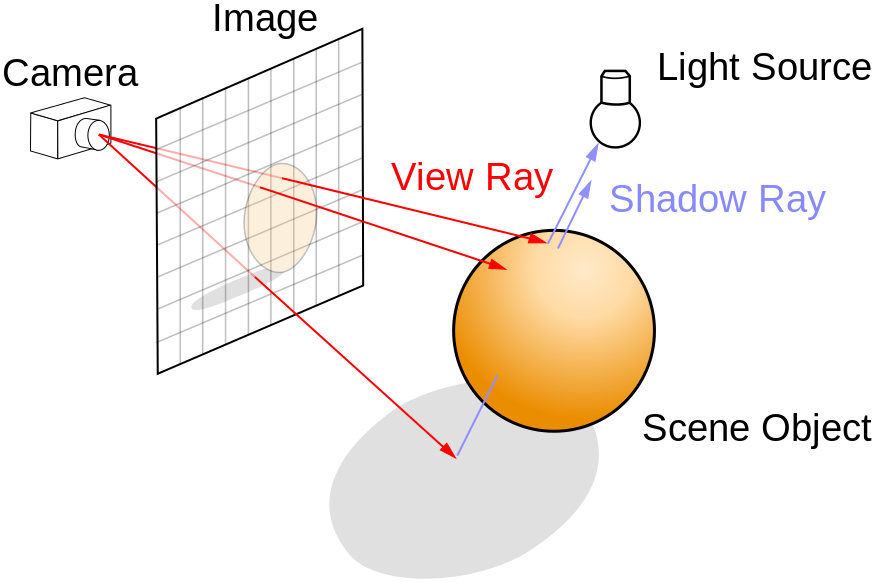
# Ray tracing algorithm

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

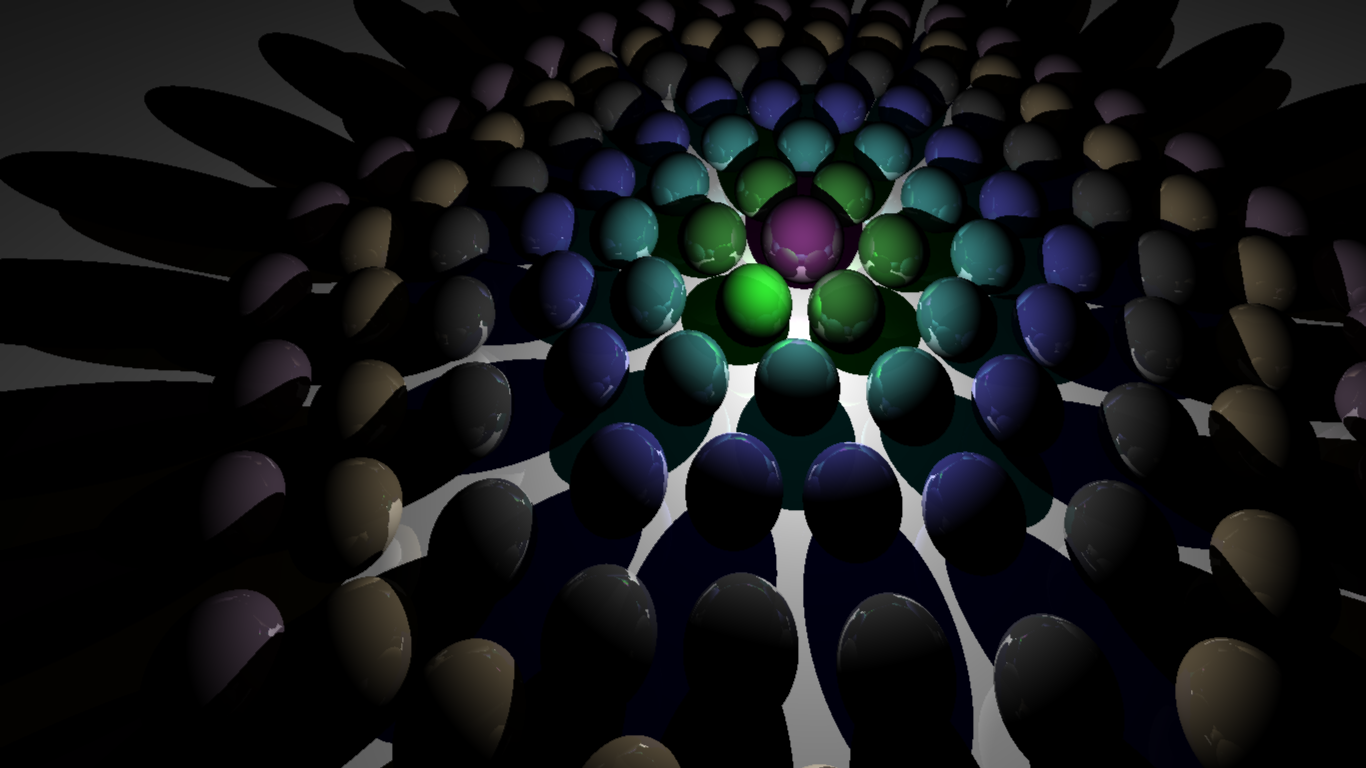
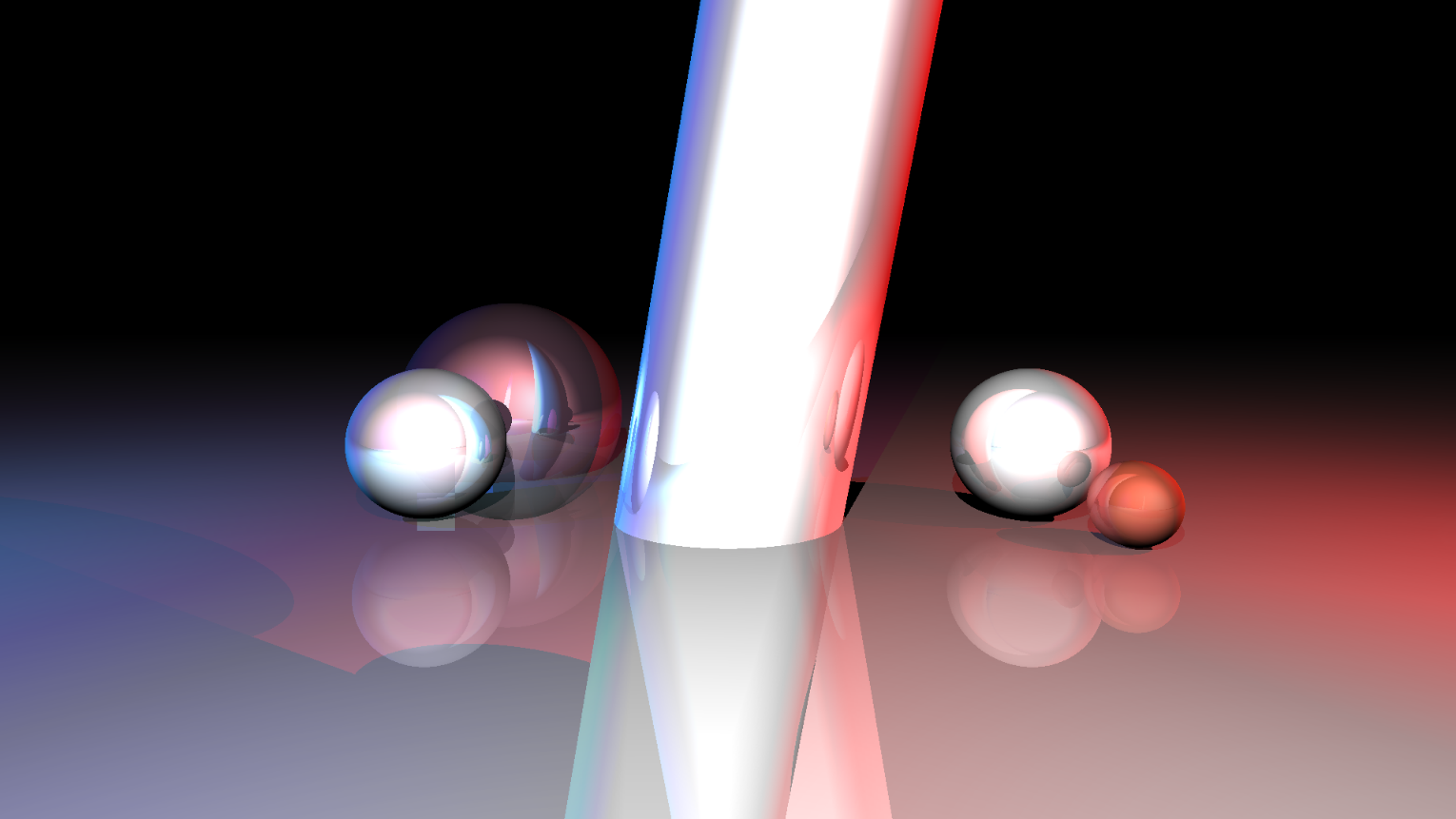
* We send a “solar ray” from the eye through each pixel of the view screen. For example, if we want to generate a 1600x900 image, we will have to send at least 1600 \* 900 = 1’440’000 rays)
* For each ray, we will check if the ray touch one of the object, using mathematical formulas. If we don’t, we 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, 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



# Compiled scenes

The scenes will be “compiled” by the client in order to optimize the transaction between the servers. The scene will be compressed to binary.

The file format is the following:

## File Header

### Module number

The number of modules to load. The order of the modules will identify them in the body of the file

### Module list

A list of names. These names have to be a sequence of 25 ASCII characters. These names are the names given by the getName() function inside one of the loaded library.

If one of the module can’t be found, the server cannot handle the scene and have to send a COMPIL\_FAIL message to the client.

### Camera description

Hold a description of the camera. The values are the position and orientation of the camera.

## File Content

Then comes a description for each Object. An Object is a structure containing:

### ID

The module index in the array given in the header of the file

### SUB-ID

The index of the instance in the library. For example, our “Basic-shape” module can be:

0: A sphere

1: A Cone

2: A Cylinder

3: A Plan

### Number of sub-modules

The number of sub-modules associated with the object

### Sub-Modules Objects

A description of an object (Id, Sub-id, number of sub-modules, Sub-modules objects)

This can describe a translation, a rotation, a light intensity, an opacity…

# Versioning

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





