VR CoralReef: Reimplementation and Enhancement of the coral reef simulation SICCOM into the agent-based modeling framework FLAME

Tobias Brandt Stefan Heitmann

ABSTRACT

This paper describes the simulation part of the coral reef visualization of the *VR CoralReef 2* project.

1. INTRODUCTION

This document serves as a technical report for the coral reef simulation plugin, developed in the master project *VR CoralReef 2* at the University of Bremen. The overall goal of the *VR CoralReef* project is it to bring attention to the increasing damage afflicted upon coral reefs all over the world. *VR CoralReef 2* is a follow-up project of *VR CoralReef 1*, which itself is based on the 2D-simulation software *Spatial Interaction in Coral Reef Communities - SICCOM* developed by the *Leibniz-Zentrum fuer Marine Tropenforschung* in the Java programming language.

In the former project the communication exchange between the Java simulation and the visualization written in C++ was implemented via a socket connection. Therefore, the simulation needed to be simultaneously running in a seperate thread, hosted by the *Java Virtual Machine*. This lead to an resource overhead, which is not necessary. Furthermore, the amount of data that could be transferred to the visualization was limited by the socket connection. Thus only a coral reef with narrow size could be simulated and transmitted to the visualization. Since one of the focuses of *VR CoralReef 2* lays on interaction, the former socket implementation would serve as a hindrance when adding various interaction forms.

One of the main goals of *VR Coral Reef 2* was the portation of the existing project into the *Unreal Engine*. Games created in *Unreal* are mainly written in C++. Thus the idea emerged to reimplement the Java simulation in C++ in order to embed the simulation directly into the visualization as a plugin. Incorperating the simulation in this way would also remove all aforementioned problems arising from the socket implementation.

The 2D-simulation uses the agent-based modelling framework *MASON* [2]. To stay as close to the original implementation as possible, the reimplementation should utilize an agent-based modelling framework as well. Further, the framework should provide the capability to simulate large populations. Therefore, the C-based *Flexible Large-scale Agent Modelling Environment*, short: *FLAME*, was chosen.

2. PREVIOUS WORK

As mentioned before, the *VR CoralReef 2* project is based on the work of the *VR CoralReef* project at the University of Bremen [1], which itself gathered the coral simulation data according to previous work by the ZMT [6]. Since then, further research has been done to study the effects on artificial disturbances [5].
The implementation which is described in this paper, uses the agend-based modelling software *FLAME*, which is further explained in [3] and [4]. Furthermore, there are extensions, which utilize the GPU to simulate large amouts of agents in parallel [7].

3. METHODS

The plugin consists of three distinct modules, separated by their respective responsibilities: a simulation module, a processing module and a communication module. Each of the submodules is further explained in its corresponding subchapter. Figure 3 gives an overview over the whole plugin.

The plugin itself is depicted using the dashed box at the bottom and contains the simulation module as well as the processing module. It communicates with the visualization through the communication module inside the visualization itself, depicted in the dashed box at the top.
The simulation module encompasses the calculation of coral data due to agent-based simulation of corals, algae and certain utility agents, such as a temperature agent (see 3.1). It interacts with the processing module by sending over relevant agent data at the end of each simulation iteration, which represents one specific point in time. User interaction is mainly provided via manipulation of global simulation states by the processing module.
The processing module (see 3.2) serves as an interlayer between the agent-based simulation in C and the communication module to the visualization in C++. It processes all accepted data of an iteration into a frame, representing the state of the reef at that specific point in time. The frame is then sent to the communication module. Moreover, the processing module enables user interaction by providing an

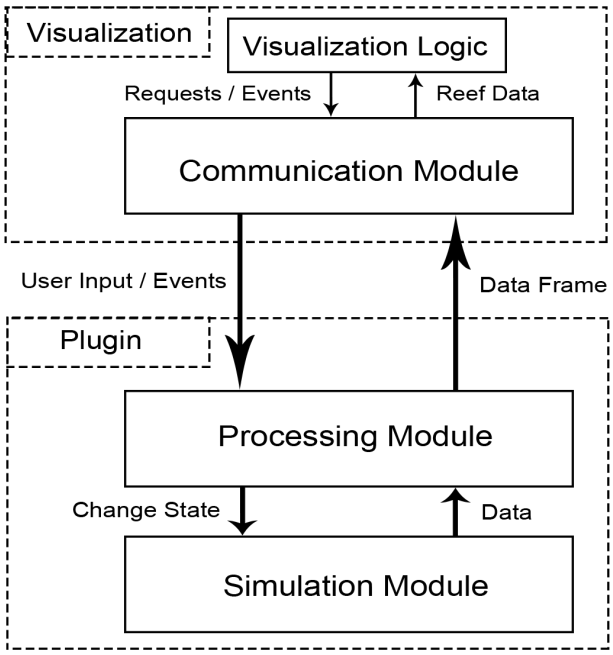

Figure 1: Overview of the Plugin Modules

interface through which the communication module can manipulate the simulation.
The communication module (see 3.3) communicates between plugin and visualization. It accepts coral data frames from the processing module, processes them into a suitable representation for further usage and passes them to the visualization. Additionally, it acts as an interface for interaction with the simulation plugin.

3.1 Simulation module

The simulation is entirely achieved through simultaneous execution of a large amount of independent agents. An agent is a finite-state machine (FSM) consisting of states and transitions. Each agent is defined via a model written in XML. A model includes the name of an agent, its attributes and its states as well as its transition names. Each transition name is referring to a C-function, which can operate on global and the attributes of the agent and is called when transitioning between states.
Models are also able to define additional environment variables. The values for this variables are defined through an environment file called *0.xml*. This file serves as an initial input file for the simulation representing its initial state. After each simulation cycle a new environment file is created, containing the updated state of the reef, which subsequently serves as the new input state.
Since coral reefs consist of numerous interdependent individuals, there is a need for communication between agents. Hence agents are able to communicate via message boards. Message boards are defined by their message, which can be a part of a model as well. A message consists of its name and various custom message attributes.
Due to the interaction between agents, a problem arises: An agent's state transition might require information sent by another agent and might depend on another agent's state transition to be run in prior. Thus state transitions are automatically assigned to a respective execution layer inside the execution order by *FLAME*.

Figure 3.1 shows a simulation cycle of a *Porites Lobata* coral. At the beginning of a cycle the singleton lobata group agent may create (*recruit*) multiple new lobata agents with random positions and sizes. Thereafter the lobata agent checks if it is still in the bounds of the simulated coral reef. If it is not, the lobata *dies* and sends a *death message* containing its last state to the singleton data collector agent, before stopping its execution. If the lobata agent is still within the bounds, it receives its probabilities of coral bleaching and dying from the singleton temperature agent. It then utilizes those probabilities to calcutate its grade of bleaching. The lobata may die during this phase. Hence, the lobata may send a death message to the data collector.
Consequently, the lobata agent adjusts its attributes for growth and its radius, before sending its updated state to all other corals for them to calculate their subsequent interaction with the lobata. In turn the lobata receives the states of all other corals and begin to update its state as result of interacting with neighboring corals. Analogously to the bounds check and bleaching, the lobata may die due to the interaction. In this case the lobata would send over its *death message* analogously.
After applying the previously calculated growth attributes and updating its age, the lobata passes a message containing it's surface to its coral group, the lobata group. The lobata group utilizes this information for further lobata recruitment.
The last step inside the simulation cycle incorporates the lobata sending its state to the singleton data collector agent. In contrast to the aforementioned *death messages*, this messages illustrates that the lobata is still alive after the simulation step. The data collector agent is part of the *processing module* described in the next subchapter.

3.2 Processing module

As stated before, the processing module acts as an interlayer between the C simulation and the C++ visualization. Its main task is embedding the simulation by governing its execution utilizing functions to initialize, run and stop it as well as passing on simulation data to the communication module for the visualization. Moreover, it deletes simulation state files which become obsolete and accepts user input and events like mechanical disturbances from the communication module.
Figure 3.2 depicts the flow of simulation data starting in the simulation module at the bottom. The data is handed upwards. Interaction between the simulation in C and the C++ interface for accessing simulation data is achieved through a buffer. The buffer stores the reef state after an iteration as a *frame* and communicates with both C and C++ via an interface to C and C++ respectively.
Simulation data is originally determined according to the aforementioned simulation process. After each simulation iteration, the data collector collects the states of all corals inside the reef. It then processes those states together with additional reef data like temperature and amount of corals into a frame representing the reef state after the iteration.
Subsequently, the simulation frame is transferred from the data collector to the frame buffer. If the frame buffer is

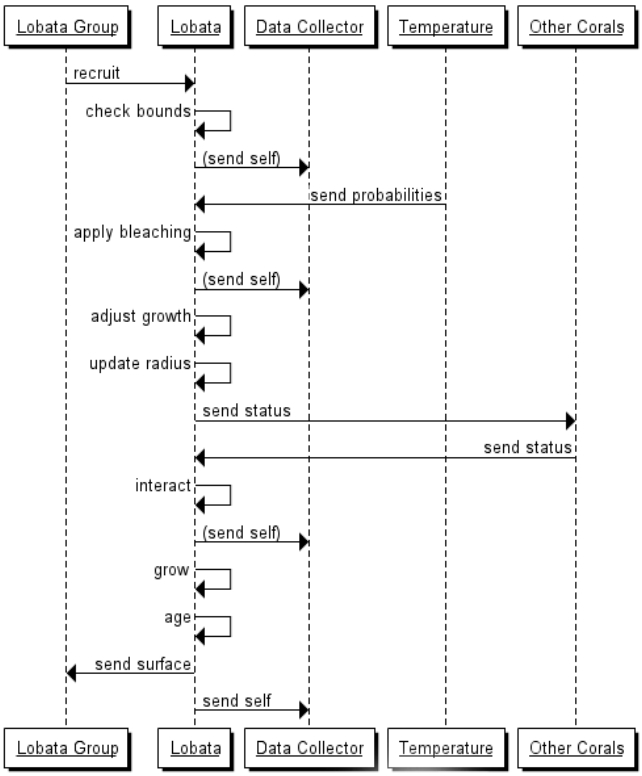

Figure 2: Lobata Agent Simulation Cycle

full, the simulation blocks and waits until a frame can be inserted. This leads to an automatic pausing and prevents that the simulated state and visualized state move too much out of synchronization.
The simulation frames inside the frame data can then be requested via an interface in C++ and, provided a frame is stored inside the frame buffer, a simulation frame is returned.

3.3 Communication module

The main purpose of the communication module is to serve as an interface between processing module and the visualization in the *Unreal Engine*. Hence, most of its functions redirect requests from the visualization to the processing module.
In regular, user-defined intervals, the visualization sends a signal to update the state of the reef. If such a signal is received and the processing module holds another processed frame, a separate thread starts processing the requested new frame from the processing module. All objects within this frame then get allocated on the heap. The thus created references get passed on to the visualization as soon as it sends a signal again. Deallocation is provided via an interface that lets the visualization deallocated coral data on demand. To efficiently manage the data allocation and deallocation process, an identifier map is employed to keep track of the allocated coral identifiers.

3.4 Disturbance Events

One event that deserves special attention and needs to get a brief explanation is the *disturbance* event. It simulates a mechanical pertubation and serves to illustrate the endangerment of coral reefs. If such an event is triggered, a signal containing the dimension of the caused event is sent to the communication module, which in turn passes it on to the processing module. Subsequently, the processing module manipulates the global simulation state in such a way that all entities within the dimensions of the disturbance get destroyed. After the disturbance has been processed inside the simulation module, the simulation sends a signal to the visualization passing through the processing and communication module. On reception of the signal the visualization invokes the visual representation of the disturbance. After the visual event has finished, the frame, in which the disturbance got invoked, is displayed by the visualization.

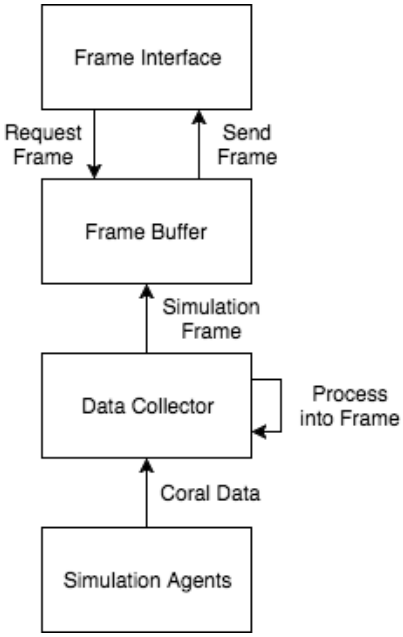

Figure 3: Simulation Frame Data Flow

4. RESULTS

The original goal of embedding the *SICCOM* simulation directly into the visualization was achieved by reimplementation using the C-framework *FLAME*. In comparison to the former implementation, this approach eliminates unnecessary overhead as well as limitations that were formerly imposed by the socket communication. Hence, it is possible to simulate and therefore visualize larger coral reefs. Despite this, the portation is currently perceived to run slower than its original implementation, due to the fact that the *MASON* framework is utilizing highly efficient geometric data structures.
The native implementation in C and the integration as an *Unreal Plugin* enables the user to interact with the simulation during runtime. The user is thereby able to directly influence the simulation by changing the water temperature, triggering external disturbances or restarting the entire simulation to experience the growth of the reef anew.
Moreover, the removal of the socket connection benefits developers, because the interface between visualization and simulation can more easily be adjusted and expanded. This

is due to the fact that there is a direct data transfer instead of an indirect one over a socket connection, also resulting in less problems in terms of synchronization.

5. FUTURE WORK

This paper provided insight into the reimplementation of the *SICCOM* simulation directly into the coral reef visualization project developed in terms of the master's project *VR Coral Reef 2*. Due to the restricted time frame of the project, there are still areas that could be improved in the future. Those include further enhancement of performance and user experience.

One possibility would be to use the framework *FLAME GPU* instead of the regular *FLAME* framework. This would vastly increase the performance due to massive parallelization on the GPU utilizing *CUDA*. Since *FLAME GPU* is an extension of *FLAME*, the portation effort of the existing agent models would be minimal.

Furthermore, additional user interaction metaphors could be integrated into the simulation. Currently, there is only one kind of disturbance event, but there are several kinds of other external disturbances not yet implemented. User defined rewinding and fast-fowarding could also be added.

6. REFERENCES

[1] An interactive 3d simulation of the life of coral reefs and its evolution depending on changing environmental parameters.

[2] G. C. Balan, C. Cioffi-Revilla, S. Luke, L. Panait, and S. Paus. MASON: A Java Multi-Agent Simulation Library. In *Proceedings of the Agent 2003 Conference*, 2003.

[3] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the flame agent-based simulation framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 538–545, June 2012.

[4] C. Greenough, S. Chin, D. Worth, S. Coakley, M. Holcombe, and M. Kiran. An approach to the parallelisation of agent-based applications. *ERCIM News*, 2010, 2010.

[5] A. Kubicek, C. Muhando, and H. Reuter. Simulations of long-term community dynamics in coral reefs - how perturbations shape trajectories. *PLOS Computational Biology*, 8(11):1–16, 11 2012.

[6] A. Kubicek and H. Reuter. Mechanics of multiple feedbacks in benthic coral reef communities. *Ecological Modelling*, 329(C):29–40, 2016.

[7] P. Richmond and D. Romano. Agent based gpu, a real-time 3d simulation and interactive visualisation framework for massive agent based modelling on the gpu. In *In Proceedings International Workshop on Super Visualisation (IWSV08) 2008. In.* Press.

Appendix

File name	Description
simulation.c	Main file for scheduling and managing the simulation
lobata.c	Implements the behavior of a lobata agent
lobata_group.c	Implements the behavior of the singleton lobata group agent
lutea.c	Implements the behavior of a lutea agent
lutea_group.c	Implements the behavior of the singleton lutea group agent
muricata.c	Implements the behavior of a muricata agent
muricata_group.c	Implements the behavior of the singleton muricata group agent
damicornis.c	Implements the behavior of a damicornis agent
damicornis_group.c	Implements the behavior of the singleton damicornis group agent
alga.c	Implements the behavior of the a agent
alga_group.c	Implements the behavior of the singleton alga group agent
turf.c	Implements the behavior of a turf cell agent
turf_group.c	Implements the behavior of the singleton turf cell group agent
temperature.c	Updates the simulation temperate and calculates coral bleaching/death probabilities
recorded_temps.h	Stores temperature samples for calculating reef temperature
utils.c	Provides utility functions, e.g. geometrical intersection/containment functions
grazing.c	Calculates the amount of grazing inside the simulated reef

Table 1: Simulation Module: Corresponding Files

File name	Description
sicPlugin.cpp	Acts as an interface to the communication module
CoralBuffer.cpp	Data structure for holding coral data frames
SharedBuffer.cpp	Manages the usage and access of the coral buffer
data_collector.c	Agent that collects and transfers information of the simulation

Table 2: Processing Module: Corresponding Files

File name	Description
PluginCommunicator.cpp	Implements the communication module

Table 3: Communication Module: Corresponding Files