Bachelor project Computing Science – Starting form

*To be filled in and signed by the student and the supervisor(s) at the beginning of the bachelor project.*

Course code: WBCS13000

Credits (EC): 15 points (420 hours)

1. **General information**

Student name and signature: Wiebe-Marten Wijnja

Student number: s2776278

Project title: Efficiently rendering Iterated Function Systems when zooming in

Date:

Starting meeting: 2020-05-08

Final assessment: *should be submitted 10 weeks after starting date*  
  
First supervisor (name + signature):  
  
Second supervisor (name + signature):

Daily supervisor (if applicable):  
  
External supervisor (if applicable):

1. **Project description:**

Iterated Function Systems (IFSs) are a method of constructing fractals [Zob92]. The resulting fractals are often self-similar.

Until now, very little research has been done in efficiently rendering IFS-generated fractals when altering the camera angle that looks at such a fractal, such as when zooming in. The contractive and self-similar natures of IFSs might provide with some interesting opportunities to improve their rendering process.

In this project, the student will investigate investigate this and come up with new ways to speed up the rendering of IFSs, in the hopes of creating an algorithm that runs fast enough on modern consumer hardware to render IFSs in high detail in real-time.

The research question: “**Is it possible to efficiently render animations in which a camera zooms in on an Iterated Function System fractal in real-time?**”

The most commonly-used algorithm to do IFS-rendering is the ‘chaos game’ [Bar88]. This is a non-deterministic algorithm that converges to a deterministic result because of the contractive nature of IFSs. Because of its simplicity, it can run very fast. However, the chaos game is usually used in a way in which we draw immediately pixels on a canvas, which means that no calculations can be re-used between to-be-rendered animation frames.

Another disadvantage of the chaos game is that it is not trivial to use on current GPU (OpenGL) pipelines. Although some prior work that uses GPGPU techniques to optimize the chaos game algorithm exists [Gre05], this still leaves room for further improvement.

**References:**

[Zob92] Zobrist, George Winston, and Chaman Sabharwal, eds. *Progress in Computer Graphics.* Vol. 1. Intellect Books, 1992. (p. 135-141)

[Bar88] Barnsley, Michael F. *Fractals everywhere*. Academic press, 2014.

[Gre05] Green, Simon G. "GPU-accelerated iterated function systems." *ACM SIGGRAPH 2005 Sketches*. 2005. 15-es.

1. **Methodology and timeline:**

To create an algorithm that works efficiently with graphics systems, a language environment should be used that:

* compiles directly to machine code and does not doe garbage collection, which will make it easier to benchmark rendering speeds.
* is able to communicate well with a GPU, which will be paramount in speeding up the rendering process.

**Timeline:**

|  |  |  |
| --- | --- | --- |
| **Week** | **Work** | **Milestones** |
| 1 | Literature study | Come up with ways to improve IFS rendering |
| 2 |
| 3 | Design and Prototype | Create a prototype to validate these new rendering ideas |
| 4 |
| 5 | Refining the program, improving efficiency | Improve the prototype to make it more efficient and easier to use/understand |
| 6 |
| 7 |
| 8 |
| 9 | Writing and finalizing Thesis |  |
| 10 |

1. **Division of tasks**

This project will be done by a single student.

1. **Deliverables:**

* BSc thesis;
* Software source code;
* Source code documentation;
* Example images/animations of generated IFS-fractals.

1. **Grading**

*This section should briefly describe how the grade will be composed by summing up weighted contributions on the following topics:*

*Scientific quality of Research and technical contribution: 40%*

*Project management and interpersonal skills: 20%*

*Final Presentation: 20%*

*Report/Thesis: 20%*

*The contribution of all topics should all have a non-zero weight, since they are mandatory parts. All weights should add up to 100%.*