Technical Report

Extrapolate and Conquer

TSBK03 HT 2013

Version 0.1

2013–12–23

Extrapolate and Conquer

Teknik för avancerade datorspel, HT 2013
Department of Electrical Engineering (ISY), Linköping University

Participants

Name	Tag	Responsibilities	Phone	E-mail
Gustav Häger	GH		070–649 03 97	gusha124@student.liu.se
Alexander Sjöholm	AS		076–225 11 74	alesj050@student.liu.se
Mattias Tiger	MT		073–695 71 53	matti166@student.liu.se

Examiner: Ingemar Ragnemalm, ingis@isy.liu.se

Contents

List of Figures

List of Tables

1 Introduction

The aim of this project was to develop a computer graphics application combining several state-of-the-art techniques into one beautiful and intelligent world.

2 System Core

We built and uses an Entity System as an underlying game engine framework. The idea behind an entity system is that objects should be treated as pure aggregations of data containers, with game logic being separated from objects all together. Instead of having deep class hierarchies and chained method calls, logic for managing specific components is batched on all such components in the system. This provides some advantages over other approaches such that the architecture becomes more flexible and extendable. It is clearer how to build functionality and especially where. Another advantage from the batching is the possibility of much higher performance (maximizing caching and minimizing cache misses) as well as simplifying parallelism of logic processing.

2.1 Entity System

An Entity System consists of three main parts: Entities, Components and Systems. An Entity is simply a label or identifier of an object. A Component is a pure data containers, and each entity has a collection of none to several different components. A System consist of logic for working with primarily one, but sometimes several, components. So, an object is an entity label and a collection of components that belong to it. The object is updated by different systems performing tasks on the components. A example used in this project is seen in figure 2.1.

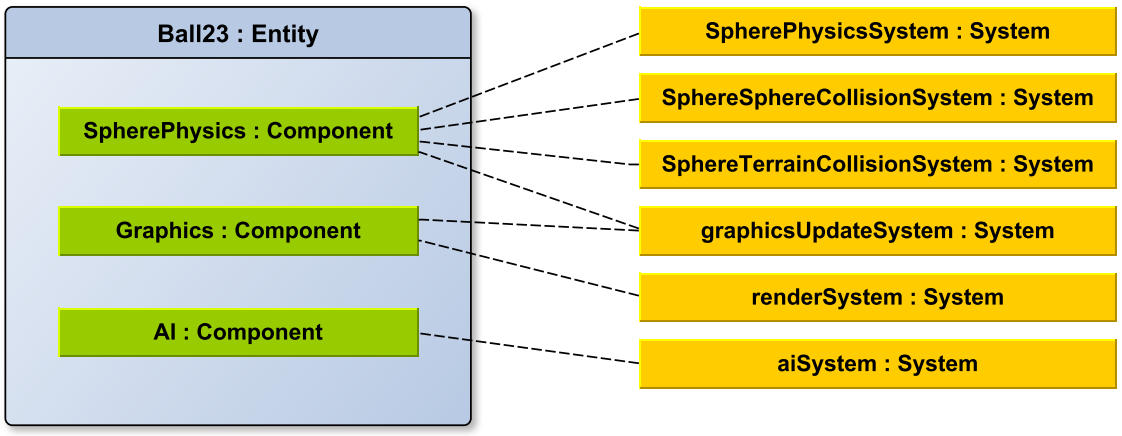


Figure 2.1: Caption

(TODO: reformulera och skriv nedan mer flytande och omarbetat)
* The idea behind: Systems perform on ALL components of a certain type, which is fetched from the EntityManager.
* Our system allows for components to require other components. This means that systems which work on multiple components can be proven to always work by fetching the component which require the others. The require check is performed at compile time and code is generated to handle the specific requirement-tree specified. The program writes part of itself to be maximum efficient and robust, by specifications by the developer in the source code.
* Easy to use (a natural work flow of what goes where and how to solve problems. minimal overhead to use the system), easy to maintain, easy to extend, extremely efficient, trivial to parallelize calculations, verifies consistency at compile time...

2.2 Rendering

2.3 Physics

3 Graphics

Graphics graphics.

3.1 Generating a World

How to world

A world is easily divided into different aspects. There are the sky, the oceans and the land. The land contain different terrain, with different types of ground and vegetation. There is a sun orbiting the world, casting rays of light and in the process creating reflections and indirectly making shadows.

A procedural world is generated by carefully chosen algorithms. Our world is procedurally generated anew in a new unique constellation on every run, or a seed can be provided to generate specific worlds. The terrain is first formed, then the ground texturing and the vegetation, both dependent on properties of the terrain. The shadows depend on the sun and the waves on the terrain as well as on time itself.

3.1.1 Sky

The sky is achieved using a high-resolution texture of a sky mapped to a skybox. The mathematical location of the sun is placed as close as possible to the sun appearing in this texture. This gives shadows and shading a natural feel. The texture has been manually modified at the horizon to fade towards a shadowish gray color. The color is the same as the one objects are distance-fogged with. This makes the sky melt into the ocean in a very nice way.

3.1.2 Ocean

// TODO Tiger
Normal-mapped square with moving normal map. Makes the water glister/glitter from a far.

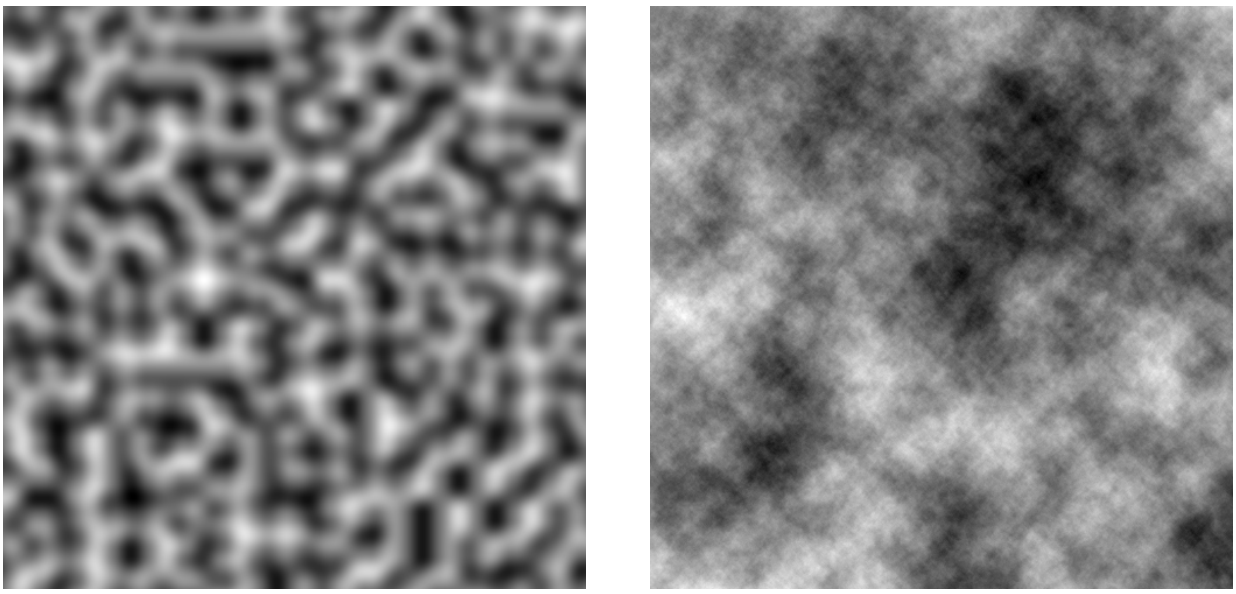
FIGURE
Waves on the beaches is made by having several sinus waves aggregate horizontally to vary the wave fronts, and by having a sinus wave that control the vertical assent/decent of the waves.

FIGURE

3.1.3 Terrain

The terrain is generated by sampling a noise function and translating its value into a height for the current vertex. The noise in this case originates from a Simplex function. However, to get a realistically looking terrain one it is not sufficient to sample this function only once for every vertex.

Fractional Brownian Motion is calculated by sampling the Simplex function at different frequencies and calculating a weighted sum over the samples [1]. The result is a nice looking height map.



(a) Height map generated from single-octave simplex noise (b) Height map generated with Fractional Brownian Motion

Figure 3.1: *Comparison of noise functions*

3.1.4 Ground

The ground is textured using a non-linear multi-texturing approach based on both altitude and terrain slope. Currently only three textures are used, one sandy, one grassy and one rocky. ... TODO: blah...

FIGURE
FIGURE

3.1.5 Content

// TODO - Häger
Trees and rocks

3.2 Visual Effects

Boom hacka lacka

3.2.1 Shadows

Shadows are one those things that can make a scene really come alive. There are many techniques in which shadows can be achieved with different pros and cons. We have chosen to use shadow mapping since it offers real-time shadows for arbitrary shapes in a theoretically straight forward way. The quality of these shadows can be increased arbitrarily, but the computational complexity is increased equivalently, which is what limits this method.

More specifically our implementation utilizes *Light-Space Shadow Mapping* which can be summarized in the following steps:

- Place the camera in the light source and adjust the camera frustum to cover the part of the scene that will be visible in the final render.
- Render the scene with as simple shaders as possible and store the depth buffer. This is the shadow map.
- Place the camera in its final-render location.
- For each vertex:
  - Transform into light-space coordinates.
  - Compare the distance from the light source with the corresponding value in the shadow map.
  - If the vertex is further away than the shadow map indicates, it will be shadowed.

3.2.2 Fog

The transition between different parts of the world can sometimes be very sharp in an unpleasant way. For instance, at the border between the sky and the ocean seen in figure //TODO below.

// FIGURE

This can be remedied be adding some distance-fog to the ocean. If the color of the fog matches the color of the skybox at its horizon the transition will be seamless. Our skybox has been modify to fade into the color of the fog at its horizon, which can be seen in figure //TODO below. Notice that we have chosen to not let the skybox be affected by any fog. By doing so one can always see the sky when looking up, which is rather pleasant.

// FIGURE

The distance-fog is also good for constraining the rendering size of the current scene. By adjusting the distance at which the fog appears one can adjust how much that is needed to be rendered of the scene. This enables an arbitrarily large world to be present without killing your computer since only the visible part of the world inside fog-limit needs to be rendered.

3.2.3 Normal Mapping

Normal mapping is a technique for adding fine details to an object without adding more vertices, which saves a lot of geometry computations. A normal map is generally an image where the RGB-channels represent x,y and z coordinates for a normal vector. This texture is used in the fragment shader when computing the shading for the current fragment. Before calculating the shading the normal vector is rotated to match the object on which it is to be mapped. Notice that the normal vector is not translated since we want it to remain as an normalized direction and nothing more. An example of a normal map can be seen in figure //TODO below.

// FIGURE

4 Artificial Intelligence

Ai ai ai ai ai ai ai ai.

5 Conclusions

Awesome

References

[1] Mandelbrot, B. & Van Ness, J. (1968)
"*Fractional Brownian Motions, Fractional Noises and Applications*"
SIAM Review, Vol. 10, No. 4, pp. 422-437

[2]

[3]

[4] Gardel, A., Bravo, I., Jimenez, P., Lazaro, J.L. & Torquemada, A.
"*Statistical Background Models with Shadow Detection for Video Based Tracking*,"
Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on?? Page: 1-6.

[5] Zivkovic, Z. & Heijden, F.
"*Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction*,"
Pattern recognition letters, Vol. 27, No. 7. (2006), pp. 773-780.

[6] Bernardin, K. & Stiefelhagen, R (2008)
"*Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics*,"
Interactive Systems Lab, Institut für Theoretische Informatik,
Universität Karlsruhe, 76131 Karlsruhe, Germany

[7] "*CAVIAR: Context Aware Vision using Image-based Active Recognition*,"
EC Funded CAVIAR project/IST 2001 37540
http://homepages.inf.ed.ac.uk/rbf/CAVIAR/

[8] Hirschmüller, H (2008)
"*Stereo Processing by Semiglobal Matching and Mutual Information*,"
IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 30(2) pp. 328-341.

[9] OpenCV *Open source computer vision*
http://docs.opencv.org/
Accessed on 2013-12-13