Major Project Report

Tom Minor - Level H
Major Project
Bournemouth University - NCCA

March 9, 2016

Project Overview

Main Aspects

1. Fractal Renderer
2. FX Work
3. Pipeline

Initial Research

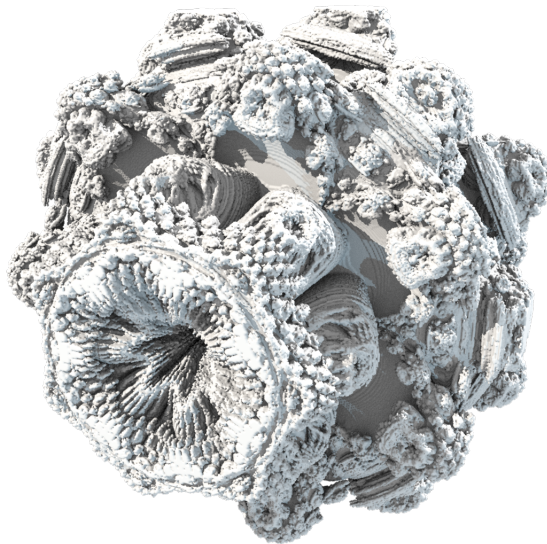## 1.1 Trying out existing stuff

1.1.1 Fractals in FX Software

Looked at developing fractals in Houdini, may not need to develop a custom tool to visualise

Volumetric mandelbulb

Voxel Based

Slow to compute

Disney used houdini on BH6, but it's much too slow for our needs.

Need a custom tool

Houdini is still good for creature effects though, example noise shader goes here

1.2 Reading the documentation and tutorials

Pipeline

- Dropbox isn't enough

- Need to version

- More effort at beginning of the project, having backups of every asset could save our asses in the end

- use perforce

- Other scripts can be added to the main pipeline suite

- Referencing pipeline required custom hooks that modify all referenced paths to be relative to $CONTACTROOT, automated, idiot proof

- Referencing in 10 files and manually clicking student popup is tedious, on save and submit remove student/education flags

- Forces the team to think in a collaborative way, they literally cannot work on the same asset at the same time

- Took some getting used to, but now the artists are used to version control and it's benefits even if it's tedious at times

- Main development time spent on PySide Qt GUI stuff, after the initial time spent learning the P4 API and commands

- Technically cross application, the P4 and GUI side of things will work whereever pyside is available. Needs a few app specific tweaks such as file saving commands etc to work properly but wouldn't take long to port

- Various wizards to automate asset/shot/lookdev file structure generation because kyran made the layout super complex

-

Renderer

- How best to make this artist friendly (Nodegraph)

- What technique (ray marching)

- Use Optix because it's faster than what i can do

- Nodegraph needs runtime compilation, at least for geo

- Does require a little hacking to get runtime code generation

- Use NVRTC to compile code at runtime and plug into preexisting functions in the optix code

- Use hacky system commands to call nvcc directly if using CUDA 6.5 or less, aka, the uni systems

- Initial dev time for node graph, runtime compilation etc is long, but in theory will allow for rapid iteration once it works

- Everything is based on demo scene stuff, can use shadertoy as reference for loads of effects

Conclusion