Major Project Report

Tom Minor - Level H
Major Project
Bournemouth University, NCCA

Supervised by Oleg Fryazinov, Adam Redford

Project Overview and Responsibilities

1. Fractal Renderer
2. FX
3. Pipeline

CONTACT is a near 3 minute long VFX sci-fi short, showing an astronaut's state of mental decay after experiencing an encounter with a 5th dimensional being while in orbit. The team worked hard to create over 80 CG assets, 3 digital environments, and a bespoke fractal render engine for the evolving tunnel sequence at the height of the piece. Over 26 shots were composited into these built environments and costumes, fleshing out the narrative and blending together the live-action and the digital elements of the film.

Chapter 1

Introduction

- Created the initial pitch idea alongside Kyran Bishop, the result was his Space Odyssey inspired story that culminated in a dramatic psychedelic sequence, where my fractals were to be used.
- I always planned to create some form of fractal lookdev tool, by joining a team of artists I made my overall job harder (the result absolutely has to look professional quality and needs to be finished or else the piece will suffer) but provided such a tool with some context.
- I took up additional roles as Pipeline TD and FX TD, given the large amount of work the team had to do it was unrealistic to expect myself to just work purely on the renderer.
- ¡Explain report overview¿

1.1 Goals

Chapter 2

Initial Research

2.0.1 Fractal Sequence

2.0.2 FX Shots

2.1 Trying out existing stuff

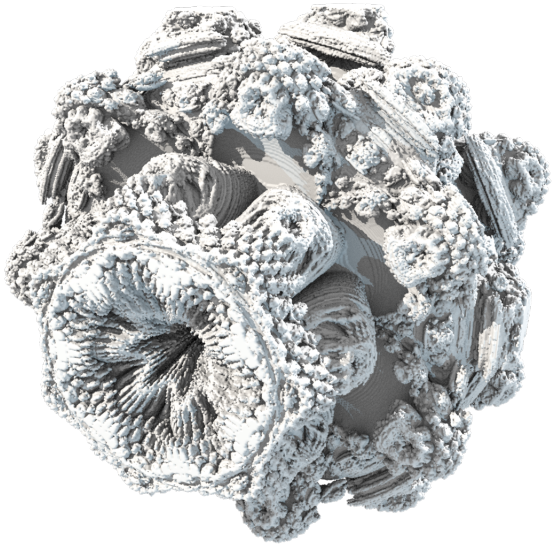
2.1.1 Fractals in FX Software

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
Shadertoy has been a big deal in the past few years
GPU Accelerated rendering of implicit surfaces really efficient
Why not develop a custom tool just for fractal lookdev

2.2 Reading the documentation and tutorials

Chapter 3

Dynamics FX

Had to create the effect of debris slowly drifting through space, so created the genesis of the destruction sequence as a base of the effect.

3.1 Setting up the simulation

- Import the Voyager model as an FBX (with UVs) and convert the Maya hierarchy setup to Houdini groups
- Fracture a slightly pre-deformed version of the mesh, transfer the UVs and create new ones in the gaps
- Generate constraints between all nearby fractured pieces
- Manually paint on the strengths of each fractured piece, this is done by painting the strength onto the vertices and then finding the average strength of each fractured piece's vertices
- The core unit of voyager has a very high strength so it does not fall apart, as it has to be intact for the shot
- The manually painted strength values are multiplied by noise to get more variation
- The strength values are remapped using a curve for a less linear falloff
- Finally, transfer the strength values to the constraints and feed into the simulation

3.2 Simulation Elements

- Fractured voyager with constraints
- Phantom collision geometry that initiates the destruction effect

- Zero gravity simulation, with torque and spin POP nodes to introduce interesting rotations into the simulation.

3.3 Final Touches

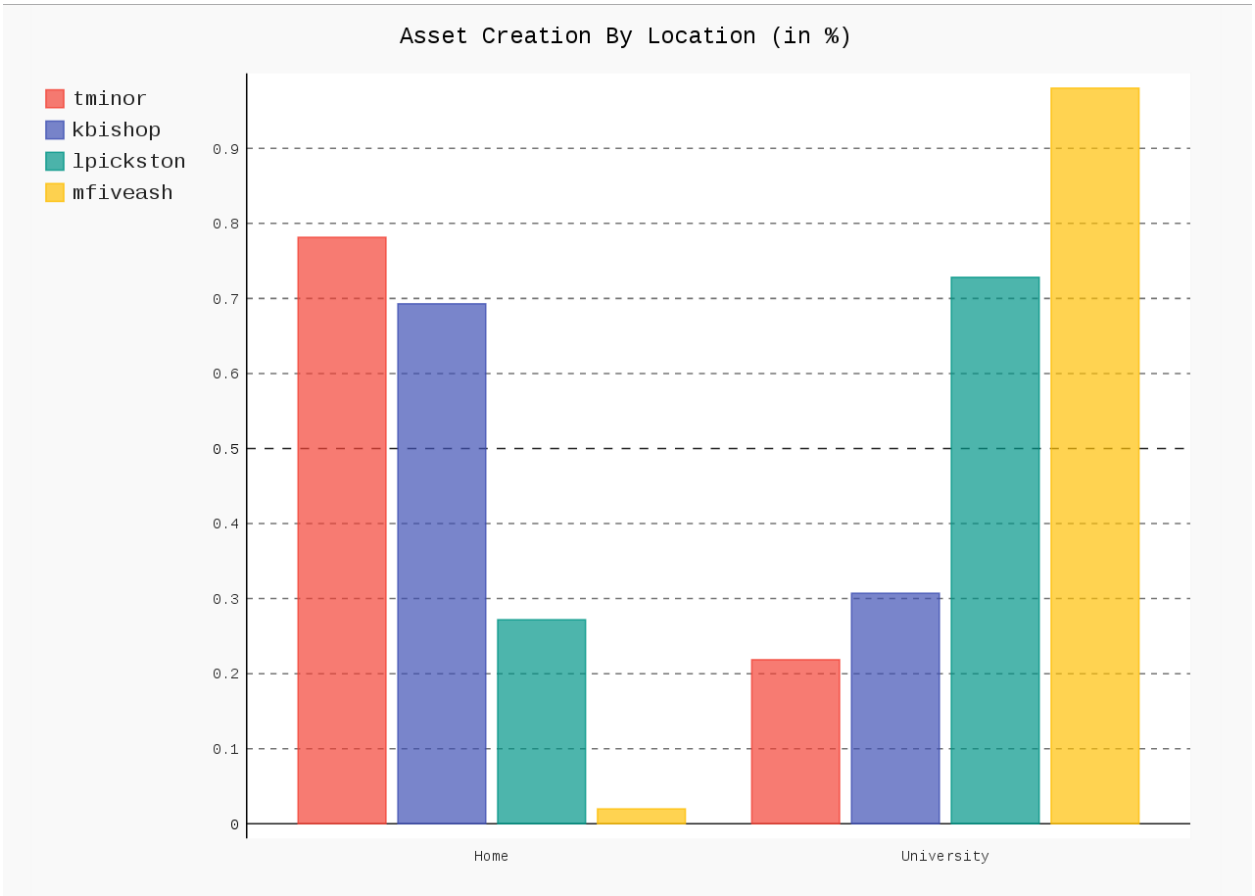
In the end, the simulation was exported to an alembic cache and Lewis processed it with a simple script that baked the animation data into locators. Using this method, we preserved animation ability for individual pieces if necessary but otherwise the simulation looked identical. Slowing down the simulation 20x in the Alembic's time scale was sufficient to give a convincing effect that the debris was slowly drifting, without completely stopping the simulation.

Chapter 4

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

- Allows us to analyse trends
- I setup Zync on Linux



Chapter 5

Fractal Sequence

5.1 Fractal Tool Requirements

- 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

5.2 Scene Management - Code Reflection

Need to change scene based on node graph

5.2.1 PTX Patching

– Initial attempt, required research into the .ptx format

- Use NVCC to compile CUDA program into ptx code and patch into the Optix ptx code, then load into Optix
- Works on my machine and university workstations, but potentially undefined behaviour and not officially supported
- Relies on my own string handling functions

5.2.2 Optix Callable Programs

- Discovered this later in the project after reading the Optix documentation fully, initially didn't notice what it was because it's not used very often compared to the other aspects of optix and isn't really clear unless you know what it does.
- Use NVCC to compile CUDA program into ptx code and then tell Optix to use this as a 'callable program', basically replacing the ptx patching process with a well defined, built in functionality.

5.3 Rendering Strategy

- Render tiles
  - For larger frame sizes this will give an opportunity for the calling program to return quickly and handle events
  - Will use less GPU RAM, which is vital for HD frames on GPUs without lots of memory
  - Has the potential to simplify implementation of other rendering algorithms in the future, for example Bidirectional Path Tracing stores light paths in an array that can become very large for huge images but is manageable for small tiles
- Monitor the Optix rendering in a separate thread and copy over to host memory every half a second or so, this keeps the GUI responsive without over saturating the PCI-E bus (GPU -¿ CPU memory transfer) with constant memory copies.

5.4 Output

- Save to EXR
- ZIP Compression, we need to render 650 1080p frames, a lot of data
- Multichannel
- Store each channel in a separate optix buffer
- Issue: Tedious to extend, but unlikely to be changed often

5.4.1 Environment Camera

- Last minute request
- Required for relighting astronaut
- Optix makes this an easy thing to change, just fire rays in such a way that all possible directions are drawn at once like a fish eye lens.
- Thankfully the way I setup the various buffers maps directly onto the new camera model with no modification, adding the camera was as simple as defining a new ray generation program. This means that EXRs can be rendered with either the pinhole or environment camera and they will have identical data passes, this allowed us to use the exact same compositing networks to calculate the environment colour and have a 1:1 match for the relighting of the astronaut. Nuke and Maya accept the generated maps as environment maps with no issues.

5.5 Colouring the fractal consistently

- Tried iteration count
- Tried comp only approaches
- Orbit traps are the way to go
- In the end we decided to use 3 different orbit traps encoded in the RGB channels of an EXR pass. This was configurable on a per scene basis if necessary (in the case of the tunnel scenes it was, the tube orbit trap gave useless values and needed manual tweaking)

5.6 How the Shots were achieved

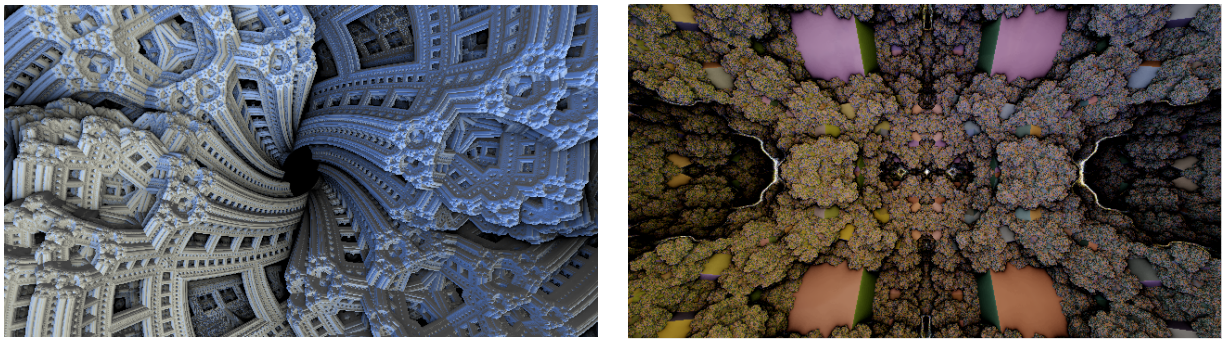
5.6.1 Side Sequence

- Use a mandelbulb, that slowly morphs from power 2 to power 5 as the sequence goes on. This produces a growing effect.
- Apply a simple scene wide transformation that moves the mandelbulb slowly along the Z axis, as well as rotates the fractal shape slowly around the Z axis.
- Set a low FOV of 30 to give the impression that the fractal shape is huge

5.6.2 Tunnel Sequence

The base shape was heavily inspired by the work of Syntopia's Shadertoy demos [3] [2].

- The main tunnel shape was originally less organic, the shape I used as reference



(a) Menger Journey [3]

(b) Kaleidoscopic Journey [2]

Figure 5.1: Two tunnel variations created by Syntopia, on Shadertoy

5.6.3 Reflected Visor Sequence

This one was done last

- Power 5 mandelbulb constrained within a manually tweaked box shape (via intersection function) to cut off the majority of it's mass and keep it's shape light and tendril like
- This entire shape is then intersected with the tunnel shape, combining the organic shape of the Mandelbulb with the hole in the tunnel. This creates an interesting silhouette.
- Primarily used as a silhouette in the comp, so not many samples were required for usable frames.
- A high FOV of 110 is used to stretch out the shape even more and ease the process of mapping it onto the visor.

Chapter 6

Future Improvements

- Look into methods besides distance estimation for more fractal possibilities, brute force approach
- Make the interface more artist friendly, with less reliance on understanding the maths
- Improve the input handling and allow for multiple input methods such as gamepads
- Some form of adaptive rendering to keep the viewport responsive
- Improve the light sampling algorithms using methods such as Bidirectional Path Tracing, VCM or Metropolis light transport.
- Provide a GUI interface for the lighting setup
- Move the batch rendering script's capabilities directly into the main renderer
- Screenspace CUDA shaders that make use of the various buffers available

Chapter 7

Conclusion

Part I

In conclusion

I believe Romanesco is a useful and novel way of developing fractal objects in a visual effects context. The majority of currently available renderers are more of a novelty, designed to create nice images for a desktop background instead of a usable plate. Attempting to use existing renderers to create fractals is a plausible method, but suffers from poor interactivity due to long calculation times for the fractal structure.

Romanesco begins to fill in the gap between the currently available approaches, with more polish and focus put towards artist usability the more important issues such as fractals not behaving well enough for real world VFX shots can be minimised by the ease in which it is possible to iterate on the fractal design using the renderer.

Despite the final result being a fairly technical tool to use, the artistic feedback from the rest of the team has been vital in getting it to this stage. If I had decided to do this project solo, I don't believe it would have developed beyond the tech demo stage. By having a strong team surrounding me, especially in the compositing department, I saw first hand which features were actually important instead of focusing on the ones that I personally had an interest in. In the end, this is the reason we ended up with 600 frames worth of fractal sequence that was successfully used in integration with a live action piece.

Chapter 8

Renderer Implementation

- Overview
- Sphere tracing distance fields
- Path Tracing
- Progressive rendering
- Tile Based Rendering
- Modifying the scene at runtime using Reflection

8.1 Performance

- Needs to be responsive, at least during the interactive preview
- Thought tile based rendering would help
-
- rtContextSetTimeoutCallback

8.2 Scene Management

8.2.1 Saving and loading scenes

- Not usable if scenes can't be saved
- Current implementation is more TD oriented, scenes are defined via CUDA functions that are compiled at runtime
- Camera information is encoded in comments at the top of the file

8.2.2 Code Reflection

Runtime compilation is necessary for allowing scene changes at runtime, specifically the ability to compile to .ptx code so it can be loaded into Optix as a callable function.

Methods

- NVRTC - Available in CUDA 6.5 and up, faster and built in way of compiling.
- System NVCC - If using an older version of CUDA (such as on the university lab workstations), I try to use the system NVCC (hopefully available in path). This has the downsides of requiring the developer tools to be installed, as well as the performance overhead of launching and managing a subprocess. However, this allows me to render on a wider range of machines regardless of the performance impact.

8.2.3 Runtime Patching

Once the code is compiled, I needed some way to tell Optix to use it.

- Manually patching the generated PTX code (prone to error)
- Optix rtCallablePrograms (officially supported way of doing it, found out later in the development process)

8.2.4 Node Graph

- More artist friendly approach
- Spent a lot of time trying to develop this in the hope that it would aid the lookdev process
- Had to abandon the concept for the sake of getting the fractals lookdev'd on time
- Influenced the current design, which is basically a pure code version of what the node graph does
- An extra level of indirection, nodes -¿ CUDA code -¿ runtime patching

Grammar Definition

In order to properly write a parser, it was important to treat the node graph like a simple language. This was helpful in figuring out error conditions, for example if a domain operation is plugged into a distance operation 8.1

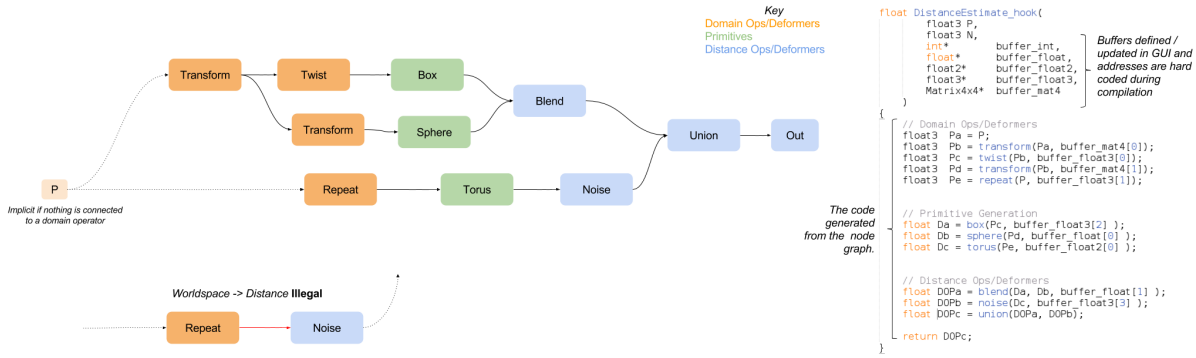


Figure 8.1: Grammar Definition

Bibliography

[1] Quilez Inigo. modeling with distance functions. http://iquilezles.org/www/articles/distfunctions/distfunctions.htm. Accessed: 20th December 2015.

[2] Syntopia. Shadertoy: Kaleidoscopic journey. https://www.shadertoy.com/view/XsX3z7. Accessed: 22th May 2016.

[3] Syntopia. Shadertoy: Menger journey. https://www.shadertoy.com/view/Mdf3z7. Accessed: 22th May 2016.