UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Whoosh Division

OcuViz - EpiUse Labs

Software Requirements Specification
and Technology Neutral Process Design

Vukile Langa	u14035449
Wynand Hugo Meiring	u13230795
Nontokozo Hlastwayo	u14414555
Gerome Schutte	u12031519

October 13, 2016

Contents

1 Vision

OcuViz is a platform for creating rich visual representations of otherwise unintuitive data by using the power of virtual reality visualisation. It aims to address the time consuming task of working with 3D scenes and make it simple, by providing a format for specifying scenes that is concise, optimised for integration with pluggable data, and to be targeted at being easily usable by anyone with even the slightest development experience.

OcuViz would typically be deployed in research, data-auditing and demonstration environments, where visualisations may be used as a medium to bring data to life, for example, describing a scene that gives viewers an interactive walk-around view of a solar cataclysm rather than simply listing numbers representing objects and scale, or allowing viewers to encounter the world from the point of view of a tiny animal, or giving demonstrators the power to transport an audience to a previously unseen location.

2 Background

EPI-Use Labs provides products aimed at getting the most out of data. The business prides itself in deploying products which allow users to sync, manipulate, extract and report various forms of data, whether through on-site or cloud-based solutions. The OcuViz project is a natural evolution of this focus. OcuViz enriches the data-reporting process for in-house developers at EPI-Use Labs and business clients alike, by adding new visual and interactive dimensions throughout their project life-cycles, whether used in product development as a tool to interact with test and debugging data, or as a value added feature in a complete solution.

3 Architecture Requirements

3.1 Access Channel Requirements

The system should be accessible via a desktop client, with mobile clients specified as optional, but most likely not feasible due to the hardware requirements of graphics processing. The target platform is Windows x64, and as such a Windows x64 client should be provided in the form of a .exe file, complete with .exe install file.

3.2 Quality Requirements

3.2.1 Performance

Stringent performance requirements are necessary, not only for novelty purposes, but due to the introduction of virtual reality. In order for visualisations running in virtual reality not to cause motion sickness in the viewer, the following minimum requirements need to be met:

- Visualisations produced must run at a constant framerate of 75fps or more.
- Overall system latency between user input and display must be kept below a maximum of 20ms.

3.2.2 Reliability

- Since input data and scene descriptor files may optionally in large part be user- or externally generated, faulty input data and scene descriptor files must be detected and reported without system crash.
- All expected user inputs for each visualisation must be clearly defined. Where an input is not declared as "expected", it must have no effect.
- The results of rendering a visualisation from a scene descriptor and input data file must be predictable and repeatable.
- Objects in a scene created via the scene editor must have the same properties in the actual visualisation as specified the scene editor.
- All visible objects specified in the scene descriptor file or placed via the scene editor must be rendered in the visualisation.

3.2.3 Scalability

- The system must be developed using technologies which are operating system neutral. Windows x64 is specified as the priority target platform, but the client may in future choose to port the system to another operating system, and must be able to do so without having to rewrite system components.

3.2.4 Flexibility

- Visualisations must be renderable using either ”hard-coded” objects fully specified in a scene descriptor file, or object blueprints specified in a scene descriptor file with ”variable” object properties that may be read from an input data file.
- Input data files must be pluggable, provided they contain valid input data as required by the object blueprints listed in the scene descriptor file.

3.2.5 Maintainability

- Version control must be used throughout the development process to create a centralised point auditing and review of code changes and simple rolling back to working system versions.
- System components must be designed to be modular, following a separation of concerns approach, in order to ease pin-pointing of system errors and swapping out of existing system modules.

3.2.6 Cost

- The client has not budgeted for additional technologies or hardware, and as such, all technologies used in development must be free for use.

3.2.7 Usability

- A complete manual must be provided describing the valid format and possible inputs of a scene descriptor file.
- Only two creation contexts for visualisations are specified: importing a scene descriptor and optional input data files, and using the scene editor. Within these contexts, each use case must have a single point of access to contribute to use cases being intuitive and predictable.
- A running visualisation must have clearly visible access to visualisation variable controls, such as viewer scale and position.

3.3 Architecture Constraints

3.3.1 Required Technologies

- **Software:**
  - Microsoft Windows 7 SP1 or newer x64 version operating system.
- **Hardware:**
  - Oculus Rift Virtual Reality Headset
  - Due to the processing requirements of virtual reality and the Oculus Rift, a host machine is required with:
    - Graphics: NVIDIA GTX 970 / AMD 290 equivalent or greater
    - Processor: Intel i5-4590 equivalent or greater
    - 8GB+ RAM
    - HDMI 1.3 video output port
    - 2x USB 3.0 ports

3.3.2 Architectural Strategies

- **Performance Strategies:**
  To meet the performance demand of graphics processing, it is suggested that an increase in processing power be used if performance doesn't meet the set requirements. Also, since graphical object creation is expensive, and often multiple similar objects are rendered in the same scene, resource re-use is to be implemented in the form of object pooling and caching.
- **Reliability Strategies:**
  In order to maintain reliable system behaviour, fault detection mechanisms must be put in place. Exception communication is to be used throughout the rendering process in the event of invalid input, with self-testing service providers ensuring the validity of input arguments and generated output. System based faults must be prevented by the use of a testing framework and test-driven development approaches.
- **Flexibility Strategies:**
  Multiple forms of flexibility strategies must be employed:
  - Process Flexibility:
    - A dedicated workflow controller must be responsible for overseeing the entire object parsing process.
    - Each system module must employ responsibility localization.
  - Service Provider Flexibility:
    - System modules and service providers must be contracts based.
    - Objects of various predefined types must be created using abstract factories.

    * Dependency injection must be employed across the system.
- Flexibility Support:
    * Automated builds must be run on each code repository change using a continuous integration server.
    * Automated testing of all use cases must be run on each automated build.