UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Whoosh Division

OcuViz - EpiUse Labs

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

September 30, 2016

Contents

1 Introduction

When presented with fairly large or small numbers we as human beings tend to struggle. One struggles to grasp the sheer magnitude of how much a trillion dollars is for example or the distance light travels in a second.

The aim of OcuViz is to create a platform for the visualization of data in 3D space using virtual reality to provide an immersive and presence depth.

2 General Information

2.1 System Overview

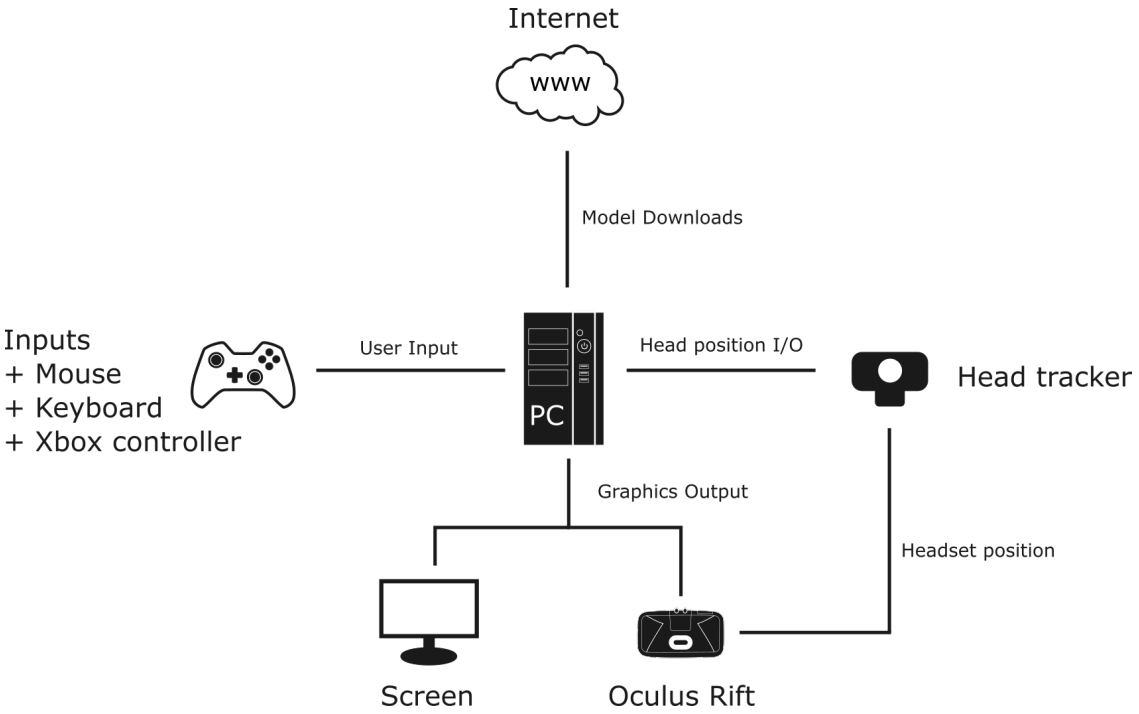
OcuViz is a platform for the visualisation of data in a virtual 3D space to allow users to conceptualise large numbers more accurately and naturally. Leveraging the power of VR we are able to create awe-inspiring and immersive scenes for users to experience. As well as, allowing users to create their own scenes through either modular CSV files which are interpreted into 3D scenes or a simplified editor without requiring the user having to be a graphics experts.

2.2 System Configuration

Minimum hardware requirements to run OcuViz

- VR Headset: Oculus Rift DK2
- GPU: GTX 970 or AMD R9 290
- CPU: Intel Core i5-4590
- RAM: 8GB
- USB: 2 USB 3.0 ports
- OS: Windows 7 SP1

Communication diagram



OcuViz receives input from the user through either keyboard, mouse and xbox controller. Or from the head tracker camera which sends location based input to OcuViz with regards to user height, head orientation, etc.

OcuViz is able to retrieve models for scenes from either local storage or via the internet where it can download specified models.

Finally graphics output is pushed to both the screen and the Oculus Rift headset.

2.2.1 Installation

OcuViz Prerequisite

- Oculus Rift Runtime Setup - https://www3.oculus.com/en-us/setup/

OcuViz will be provided in both portable and installer formats. To install simply unzip the files (portable format) or follow in the installation instructions.

3 Getting Started

OcuViz is straight forward to use. Ensure your Oculus Rift Headset, controller and head tracker are plugged in. Then launch Oculus Store. Ensure that your headset is showing a blue light. If not check Oculus Store to see the issues that exist. Also accept the health warning message displayed by the h Finally simply run OcuViz.exe to use the software. No login is required and would not make sense for this system.

Launching OcuViz

OcuViz aims to be a user friendly experience. To start using OcuViz is a straight forward procedure.

1. Ensure Oculus Rift headset, head tracker and controller (or keyboard and mouse) is plugged in.
2. Launch Oculus Rift Runtime.
3. Check that Oculus Rift headset is showing a blue light.
4. Accept the health warning message if one is displayed on the headset.
5. Run OcuViz.exe

4 Using the system

There are multiple ways to use the system. Upon startup, the user would have to use the menu scene to select one of the following:

- A predefined scene
- Use the editor to create a scene
- Select a recently created scene or
- Specify a directory containing a CSV file which will describe the scene's content.

4.1 Predefined Scenes

4.2 Editor

4.3 Recent Scene

4.4 Scene Descriptor File

In this use case, the user would specify the location of the scene descriptor file containing all the data to be rendered. But before we continue, we need to introduce the concept of an Entity.

An Entity is the final object that appears in the scene. An Entity is created using a line in a CSV file (these parameters will be specified shortly). You can further add to an Entity by using other lines to change colour, add texture, or even create a collection of the Entity containing multiple copies. In order to tell OcuViz what you want to decorate, an EntityLink is used. This is the unique name given to the Entity being rendered. So decorations are applied to an Entity that contains the given EntityLink.

Now onto how to specify an Entity. You can have one of the following base Entities:

- Shape: 3D objects such as spheres and cubes
- Light: different light sources in the scene, including directional and ambient lights
- Model: 3D models such as .obj files that are created

Certain Entities such as Shapes can then be further decorated with additional properties:

- Colour: the colour of the Entity
- Texture: add a texture to an Entity.

These decorated Entities can be used as prototypes for creating a collection of Entities. There exist two kinds of Entities:

- Collection: a basic collection with predefined types
- CustomCollection: a specialised collection with only the prototype as a predetermined property.

Scene-specific properties can be altered too. These are:

- SceneName: the name of the scene
- BackgroundColour: this changes the background colour of the scene.

4.4.1 Shape

Shape,EntityLink,type,GRAVITY,mass,xlen,ylen,zlen,xpos,ypos,zpos

The above is the format required to specify a shape. Note the lack of spaces in the format. The parameters are discussed below.
Shape: This specifies the type of Entity being created. This value never changes for all shapes.
EntityLink: As discussed above, this string is used to identify the entity being created.
type: This says the type of shape being created. Types acceptable are

- plane
- cube
- sphere
- capsule
- cylinder
- quad

GRAVITY: This is a flag indicating whether the Entity should use gravity or not. Values accepted are true or false.
mass: Floating point value indicating the mass of the Entity in kilograms.
xlen: The length of the Entity in the x plane in metres.
ylen: The length of the Entity in the y plane in metres.
zlen: The length of the Entity in the z plane in metres.
xpos: The position of the Entity in the x plane (measured in metres).
ypos: The position of the Entity in the y plane (measured in metres).
zpos: The position of the Entity in the z plane (measured in metres).

4.4.2 Light

Light,EntityLink,type,#colour,xpos,ypos,zpos,range,intensity

The line above is used to create a light in the scene.

Light: This specifies the type of Entity being created. This value never changes for all lights.
EntityLink: As discussed above, this string is used to identify the entity being created.
type: The type of light being created. Possible types are

- spot
- area
- directional
- point

#colour: No, this is not a hashtag. This is the hexadecimal colour used for the light.
xpos: The source of the light in the x plane.
ypos: The source of the light in the y plane.
zpos: The source of the light in the z plane.
range: Floating point number indicating how far the light should travel.
intensity: This value between 0 and 1 indicates how strong the light is.

4.5 Model

Model,EntityLink,http://EntityFile.obj,posX,posY,posZ

The line above imports a model into the scene.

Model: This specifies the type of Entity being created. This value never changes for all models.
EntityLink: As discussed above, this string is used to identify the entity being created.
http://EntityFile.obj: Weblink containing the model.
posX: The X position of the Model.
posY: The Y position of the Model.
posZ: The Z position of the Model.

4.5.1 Colour

Colour,EntityLink,colourName,#colour

This is the format of input to change an Entity's colour.

Colour: This specifies the type of Entity being created. This value never changes for all colours.
EntityLink: As discussed above, this string is used to identify the entity being linked to and changing.
colourName: The name of the colour. If a user is not familiar with hexadecimal colours, then one of the defaults can be used:

- black
- blue
- clear
- cyan
- gray
- green
- grey
- magenta
- red
- white
- yellow

Note: these values override the predefined colour and hexadecimal value next to it.
#colour: The colour in hexadecimal. This value will be used if colourName does not match any of the values above.

4.6 Texture

Texture,EntityLink,C://EntityFile.format

The above is used to add a texture to a Shape Entity.

Texture: This specifies the type of Entity being created. This value never changes for all textures.
EntityLink: As previously discussed above, this string is used to identify the entity being linked to and changing.
C://EntityFile.format: The file path of the texture to be used. The format must be .jpg or .png.

4.6.1 Collection

Collection,EntityLink,type,dimension,posX,posY,posZ

This is used to create a simple collection of Entities.

Collection: This specifies the type of Entity collection being created. This value never changes for all standard collections.
EntityLink: As previously discussed above, this string is used to identify the entity being used as a prototype.
type: This determines the type to be created. Possible candidates are:

- stack: this will create a stack of Entities that rain down. dimesion will determine the number of Entities in the stack, and the posX, posY and posZ values will be where they fall.
- random: dimension number of Entities will be places randomly in the cubic area between the origin and the posX, posY and posZ values specified.
- row: creates one single row with all the Entities lined up. dimension determines the number of Entities to be rendered, and the positions the starting point.
- 2d: This creates a two-dimensional area containing dimension^2 Entities, using the position as a starting point.
- 3d: This creates a three-dimensional collection containing dimension^3 Entities. Position values are used as a starting point.
- dimension: see above.
- posX: see above.
- posY: see above.
- posZ: see above.

4.6.2 CustomCollection

This one works differently. Instead of having the collection data in the same file as the Scene Descriptor, the data is kept on a separate file, referred to as the Input File. So the Scene Descriptor would have the following:

CustomCollection,EntityLink,C://InputFile.csv

CustomCollection: This specifies the type of Entity collection being created. This value never changes for all custom collections.
EntityLink: As previously discussed above, this string is used to identify the entity being used as a prototype.
C://InputFile.csv: The path to the input file.

The input file then has the following format:

posX,posY,posZ,dimX,dimY,dimZ,#colour

Note there is no EntityLink, as it has already been defined. There can also be multiple lines in the input file, and each line creates an Entity.

pos: These are used to plot the point where the Entity will be placed.
dim: These are used to change the dimensions of the prototype to whatever is needed according to the axis. The value of -1 will leave them as default (the prototype's dimension will be used in that case). **#colour:** This is used to change the colour of the Entity in question. A value of null will use the default colour provided by the prototype.

4.6.3 SceneName

SceneName,name

This gives the current scene a name.

SceneName: This specifies the type of action being done. This value never changes for all scene names.
name: The name of the scene. Be as descriptive as your imagination allows.

4.6.4 BackgroundColour

BackgroundColour,colourName,#colour

This changes the colour of the background in the scene.

BackgroundColour: This specifies the type of action being done. This value never changes for all scene names.
colourName: The name of the colour. If the name is skybox, then a tropical sky will be rendered, complete with clouds and and a sun. Else the colour will be created as with Colour. **#colour:** The hexadecimal colour being added. See Colour for more info.